Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Šarc

Zasnova in razvoj spletnega pajka za analizo dinamičnih spletnih aplikacij

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Delo je pripravljeno v skladu s Pravilnikom o podeljevanju Prešernovih nagrad študentom pod mentorstvom prof. dr. Matjaža Branka Juriča

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi zasnujte spletnega pajka za analizo dinamičnih spletnih aplikacij. Preglejte obstoječe implementacije spletnih pajkov. Seznanite se z načini izvajanja dinamičnih spletnih aplikacij in z izvajalnimi strežniškimi okolji. Definirajte posamezne komponente in izdelajte implementacijo dinamičnega spletnega pajka. Prav tako predvidite način za upravljanje spletnega pajka. S pomočjo pajka izvedite analizo povezovanja dinamičnih spletnih aplikacij z zunanjimi ponudniki storitev na izbranem vzorcu 100.000 spletnih aplikacij ter predstavite rezultate in ključne ugotovitve.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Luka Šarc, z vpisno številko **63110321**, sem avtor diplomskega dela z naslovom:

Zasnova in razvoj spletnega pajka za analizo dinamičnih spletnih aplikacij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču za dano priložnost in mentorstvo pri diplomski nalogi in Mihi Nageljnu za vodenje in pomoč pri izdelavi. Zahvaljujem se vsem kolegom iz laboratorija LIIS za pomoč, nasvete in dobro vzdušje.

Hvala domačim in vsem bližnjim, ki so mi kakorkoli pomagali pri izdelavi diplomske naloge in me na kakršenkoli način podpirali v času študija. Hvala tudi vsem kolegom, s katerimi smo šli skupaj skozi to pot.

Kazalo

Seznam uporabljenih kratic

ACID	Atomicity, Consistency, Isolation and Durability
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
ASP	Active Server Pages
BASE	Basically Available, Soft state, Eventually consistent
BLOB	Binary Large Object
BSON	Binary JSON
CAP	Consistency, Availability, Partition tolerance
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DB	Database
DOM	Document Object Model
FIFO	First In, First Out
JS	JavaScript
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
MIME	Multipurpose Internet Mail Extension
MVC	Model, View, Controller
NoSQL	Not Only SQL
PB	Podatkovna baza

PHP	Hypertext Preprocessor
RAM	Random Access Memory
REST	Representational State Transfer
RIA	Rich Internet Applications
SNS	Social Networking Service
SQL	Structured Query Language
SSD	Solid-State Drive
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
V/I	Vhod/izhod
XML	Extensible Markup Language

Povzetek

Največji delež v ekosistemu spletnih aplikacij dandanes predstavljajo dinamične spletne aplikacije. Te se med seboj povezujejo v okolju spletnega brskalnika. Uporabniki se povezovanja z zunanjimi ponudniki storitev ne zavedajo, ob tem pa lahko ponudnikom nevede razkrivajo svoje podatke. V ta namen je bil v okviru diplomske naloge zasnovan in implementiran spletni pajek, ki omogoča analizo dinamičnih spletnih aplikacij. Klasični pajki so za to neustrezni, saj preiskujejo semantiko spletnih aplikacij. Naša implementacija pajka izvaja spletno aplikacijo v peskovniku znotraj navideznega spletnega brskalnika. To omogoča sledenje virom, potrebnim za izvedbo aplikacije, in zaznavanje povezovanja spletnih aplikacij. Opravili smo sprehod po 100.000 spletnih aplikacijah. Pokazalo se je, da je stopnja povezovanja visoka – v povprečju se spletna aplikacija povezuje s šestimi ponudniki storitev, s čimer smo potrdili, da predlagana rešitev omogoča učinkovito analizo aktualne problematike.

Ključne besede: spletni pajek, dinamična spletna aplikacija, zunanji ponudniki storitev, peskovnik, integracija.

Abstract

Dynamic web applications represent the largest share in web applications ecosystem. They integrate with each other in a web browser. Users are not aware of connections with third-party service providers and may be unknowingly revealing their browsing data. In this thesis, a web crawler for dynamic web application analysis was designed and implemented to address this problem. Traditional crawlers are not sufficient for described area, since their interest is in semantics of web applications. Our implementation of crawler executes web application in a sandbox within virtual web browser. This allows crawler to track resources needed for the execution and detect integration of web applications. We conducted a crawl through 100,000 web applications. The results revealed high level of web application integration. In average, a web application integrates with six third-party providers. The results confirmed that the proposed solution provides effective analysis for described problem domain.

Keywords: web crawler, dynamic web application, third-party service providers, sandbox, integration.

Poglavje 1

Uvod in motivacija

Današnji svetovni splet sestavlja ogromno število spletnih aplikacij. Spletne aplikacije postajajo čedalje bolj kompleksne in za njihovo izvajanje ni več dovolj osnovna datoteka z vsebino in nekaj prenesenih slik. Dinamične spletne aplikacije želijo uporabniku nuditi čim več. Omogočajo visok nivo interaktivnosti, veliko dinamičnost glede na uporabnikove želje in potrebe, merijo obiske, prikazujejo napredne oglase ipd. Dinamične spletne aplikacije se med seboj podpirajo, povezujejo, integrirajo druga v drugo in nudijo svoje podatke ostalim aplikacijam. Da dinamična spletna aplikacija vse to lahko podpira, za svoje potrebe izvajanja nalaga veliko število raznih virov (ang. `resources`) od različnih zunanjih ponudnikov oziroma gostiteljev virov (ang. `resource hosts`).

V diplomski nalogi smo zasnovali in implementirali spletnega pajka z namenom izvajanja analiz o povezovanju dinamičnih spletnih aplikacij z drugimi spletnimi aplikacijami in zunanjimi ponudniki. Vse pridobljene podatke želimo hraniti v podatkovni bazi in omogočiti čim boljši vpogled nad vsemi podrobnostmi in zajetimi relacijami. Želimo si izdelati aplikacijo, s katero bomo lahko nadzorovali spletnega pajka. V ta namen smo razvili upravljalno orodje, ki deluje kot spletna aplikacija. Z njim je omogočeno zaganjanje spletnega pajka, določanje parametrov njegovega delovanja, brskanje po zajetih podatkih, shranjenih v podatkovno bazo, prikaz vseh povezovanj spletnih aplikacij z zunanjimi ponudniki in ogled splošnih analiz sprehajanj. Spletni pajek ob izvajanju dodatno gradi še hierarhijo domen. To nam prek

upravljalnega orodja omogoča vpogled v vejanje uporabljenih domen v spletnih naslovih dinamičnih spletnih aplikacij.

Analiza dinamičnih spletnih aplikacij je v ekosistemu spletnih aplikacij, ki se izvajajo v uporabnikovem spletnem brskalniku, izrednega pomena. Ko uporabnik naloži spletno aplikacijo v svojem spletnem brskalniku z namenom, da si prebere na primer današnje novice, se neposredno ne zaveda, kam vse se njegova seja povezuje za ustrezen prikaz novic. Zaveda se zgolj dotične aplikacije, ki jo je prek spletnega naslova zahteval. Ne zaveda pa se, kaj vse se nalaga in izvaja v ozadju. S spletnim pajkom želimo analizirati povezovanje in nalaganje virov dinamičnih spletnih aplikacij, ki se izvajajo v spletnem brskalniku. Pri tem je najbolj kritična izvajalna koda JavaScript. Preko nje namreč obstaja možnost zajemanja uporabnikovih zasebnih podatkov, ki se je uporabnik ne zaveda. S tem podatke nevede deli zunanjim ponudnikom ter drugim spletnim aplikacijam. Zasebni podatki lahko predstavljajo klike na miški, pritiske gumbov na tipkovnici, piškotke in aplikativne podatke izvorne spletne strani.

Klasični spletni pajki se osredotočajo na semantičnost oziroma vsebino spletnih aplikacij. Zanima jih tekstovna datoteka, ki jo kot odgovor na zahtevo generirajo strežniki. Klasični pajki te datoteke večinoma analizirajo z namenom ustvarjanja iskalnih indeksov in iskanja povezav med spletnimi stranmi za realizacijo gradnje podatkovnih baz spletnih iskalnikov. Za celostno analizo spletne aplikacije, ki se izvajaja v spletnem brskalniku, pregled tekstovne datoteke zgenerirane na strani strežnika ni dovolj in analiza dinamičnih spletnih aplikacij s klasičnimi pajki zato ni možna. Zaradi tega smo se odločili zasnovati in implementirati lastnega spletnega pajka, ki dinamično spletno aplikacijo izvaja v peskovniku znotraj navideznega okolja spletnega brskalnika. Peskovnik nam omogoča sledenje vsem dogodkom, ki se odvijajo med izvajanjem dinamične spletne aplikacije, s čimer je omogočeno sledenje nalaganju vseh za aplikacijo pomembnih virov.

V nadaljnjih raziskavah želimo izvedeti, ali zunanji ponudniki prilagajajo vire glede na spletno aplikacijo, ki vir zahteva. Izvajalna koda iz zunanjih ponudnikov se zaradi tega lahko izvaja zelo različno znotraj mnogih spletnih aplikacij, ki so zahtevale enak vir. To vzbuja skrb tudi pri že omenjenem posegu v zasebnost uporabnika. S spletnim pajkom je omogočeno podrobno analiziranje povezovanja

spletnih aplikacij, na kakšen način se le-te povezujejo z zunanjimi ponudniki virov in kakšne povezave na druge spletne aplikacije vsebujejo. Z izdelavo diplomske naloge in izvedbo vzorčnega sprehoda (ang. `crawl`) nad 100.000 dinamičnimi spletnimi aplikacijami, smo pridobili vzorčno podatkovno bazo, ki nudi osnovo za vse omenjene nadaljnje raziskave.

V prvem delu diplomske naloge (Poglavje 2) je opisana zasnova spletnega pajka. Razložili smo, kaj spletni pajek je ter kaj je dinamična spletna aplikacija, kaj pomeni izvajanje take aplikacije v peskovniku, katero je glavno izvajalno okolje in zakaj, kako smo izbrali podatkovno bazo ter zasnovali podatkovni model. Na koncu poglavja smo opisali arhitekturo našega spletnega pajka. V nadaljevanju (Poglavje 3) je najprej prikazan način, ki smo ga izbrali za povezovanje s podatkovno bazo, nato pa je pojasnjeno integriranje peskovnika v izvajalno okolje. V naslednjih dveh podpoglavjih so predstavljene komponente spletnega pajka in opisano njegovo delovanje. V Poglavju 4 so opisane tehnologije, uporabljene za razvoj upravljalnega orodja za spletnega pajka, in kaj vse to orodje omogoča uporabniku. V zadnjem poglavju (Poglavje 5) so predstavljene analize, ki jih upravljalno orodje omogoča, in rezultati vzorčnega sprehoda spletnega pajka.

Poglavje 2

Zasnova spletnega pajka

Spletni pajek je program, ki sistematično preiskuje svetovni splet (ang. `World Wide Web`) z različnimi nameni. Za spletnega pajka (ang. `web crawler`) obstaja več vrst izrazov, med drugimi je mogoče zaslediti: robot (ang. `bot`), pajek (ang. `spider`), črv (ang. `worm`), sprehajalec (ang. `walker`) in pohodnik (ang. `wanderer`) [11]. Spletnih pajkov, tako komercialnih kot raziskovalnih, je ogromno. Večinoma pa njihova koda ni odprtega značaja in je o njih znanih dokaj malo podrobnosti.

V prvih dveh podpoglavjih tega poglavja so pregledane obstoječe implementacije spletnih pajkov (2.1) in opisane dinamične spletne aplikacije (2.2). Predstavljena je izbira glavnega izvajalnega okolja našega spletnega pajka (2.3) in opis zasnove podatkovne baze in podatkovnega modela (2.4). Na koncu je opisana še arhitektura našega spletnega pajka (2.5).

2.1 Pregled obstoječih implementacij spletnih pajkov

Kljub omenjenemu pomanjkanju podrobne dokumentacije o spletnih pajkih, smo vseeno zasledili nekaj člankov, ki nudijo vpogled v arhitekturo in delovanje pajkov: `Google search engine` [4], `Mercator` [11] in `UbiCrawler` [3].

V članku [4] je predstavljen Googleov prototip vsem znanega iskalnika iz leta 1998. Google uporablja spletnega pajka za preiskovanje spleta, kjer pregleduje hi-

pertekst in rezultate indeksira za potrebe iskalnika. Napisan je v programskih jezikih C ter C++ in teče na operacijskih sistemih Solaris in Linux. Na dan preišče več sto milijonov spletnih strani. Spletni pajek je porazdeljen po več sistemih. Uporabljajo t. i. `URLServer`, ki vsakemu pajku pošlje seznam naslovov. Naloga pajkov je, da te strani prenesejo in jih potem nadalje obdelajo. Pri tem indeksirajo možne iskalne zadetke, vzorčijo besede in podobno.

`Mercator` [11] je bil razvit za merjenje kvalitete iskalnikov in je implementiran v programskem jeziku Java. Izvaja naključne sprehode po spletu, tako da iz seznama semen izbere spletno stran, kjer začne preiskovanje. Naslednjo stran izbere naključno iz pridobljenih povezav na ravnokar preiskani strani, vse dokler ne pride do strani, ki ne vsebuje nobene povezave. Takrat vzame novo spletno stran iz seznama semen. Z vsako novo odkrito spletno stranjo vzporedno tudi širi seznam semen. Ti cikli se prekinjajo s ponovnim zagonom pajka po naključnem času.

`UbiCrawler` [3] je bil razvit v programskem jeziku Java z namenom pridobivanja velikih količin podatkov za namene preučevanja strukture spleta. To vključuje statistično analizo spletnih domen kot tudi ocene rangiranja spletnih strani z namenom preoblikovanja največjega italijanskega iskalnika `Arianna`. Njegova arhitektura temelji na več agentih, ki se med seboj koordinirajo tako, da vsak preišče svoj kos spleta. Spletno stran preiskujejo z iskanjem v širino (ang. `breadth-first`). V primeru, da agent najde povezavo, ki ni lokalna spletni strani, to posreduje ustreznemu agentu, ki poskrbi za postavitev novega URL-ja (`Uniform Resource Locator`) v vrsto.

Vsi trije predstavljeni primeri večinoma preiskujejo semantiko spleta. Majhno odstopanje od tega predstavlja `UbiCrawler`, ki ima nekaj več raziskovalnega prizvoka pri omenjeni statistični analizi domen. V diplomski nalogi je semantika spleta popolnoma nepomembna, saj nas zanima nekaj popolnoma drugega. V našem spletnem pajku se želimo osredotočiti predvsem na vire, ki se nalagajo v spletnih aplikacijah. Zanima nas namreč, kaj vse določena spletna aplikacija prenese iz drugih strežnikov za svoje delovanje, kateri strežniki se največ uporabljajo za prenos virov in kateri viri se največkrat prenesejo. Sočasno želimo graditi še hierarhijo domen preiskanih spletnih aplikacij.

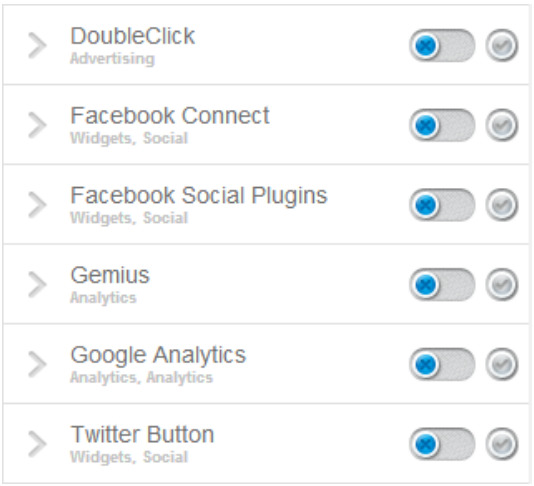
Poleg opisanih pajkov tudi večina ostalih preiskuje omenjeno vsebino spletnih

aplikacij. Klasične spletne pajke torej zanima strežniška stran spletnih aplikacij oziroma vsebina tekstovne datoteke, ki jo zgenerirajo strežniki z odgovarjanjem na zahtevo. Naš spletni pajek se od klasičnih razlikuje, saj ga zanimajo zahtevani viri, ki jih spletna aplikacija potrebuje za svoje izvajanje. Preko virov lahko ugotovimo, kako se med seboj povezujejo spletne aplikacije in od katerih zunanjih ponudnikov so odvisne. Zato naš spletni pajek preiskuje dogodke, ki se ne dogajajo na strežniku, ampak na odjemalcu. Z drugimi besedami, naš spletni pajek preiskuje odjemalsko stran spletnih aplikacij.

2.2 Dinamične spletne aplikacije

Spletna aplikacija v splošnem deluje po principu odjemalec–strežnik (ang. `client-server`), kjer je odjemalec neko orodje, ki je sposobno izvajati spletno aplikacijo (npr. spletni brskalnik). Odjemalec pošilja zahteve po virih strežnikom, ti pri sebi zgenerirajo odgovor in ga pošljejo nazaj odjemalcu. V resnici so spletne aplikacije še veliko več kot zgolj model odjemalec–strežnik in komunikacija med njima. Tudi spletni strežniki sami lahko operirajo kot odjemalci in sicer s komunikacijo z ostalimi zalednimi (ang. `back-end`) sistemi. Spletne aplikacije tako lahko opišemo kot večstopenjski sistem, pri katerem vsaka stopnja predstavlja določeno aplikacijsko plast (spletni brskalnik, spletni strežnik, aplikacijski strežnik, podatkovna baza, ipd.), vsaka plast pa lahko deluje kot odjemalec ali strežnik [19]. Spletna stran ni nujno spletna aplikacija. Kje točno je meja med njima, je težko določiti. Zasledimo lahko več razlikujočih si mnenj. Glede na definicijo spletne aplikacije bomo v tej diplomski nalogi spletno stran definirali kot vsebinsko plat oziroma kot nekaj, kar uporabnik v svojem spletnem brskalniku vidi.

Dinamična spletna aplikacija predstavlja spletno aplikacijo, ki ne vsebuje le statične vsebine, ampak se vsebina spletne aplikacije dinamično prilagaja. To je omogočeno z dinamično izvajalno kodo, ki na strežniški strani predstavlja programske jezike, kot so PHP (`Hypertext Preprocessor`), Ruby on Rails in ASP.NET (`Active Server Pages`), na strani odjemalca pa programski jezik JavaScript. Na strežniški strani omenjeni jeziki generirajo datoteko HTML (`HyperText Markup Language`), ki je prikazana v spletnem brskalniku. V odjemalcu koda JavaScript



Slika 2.1: Primer naloženih spletnih aplikacij na eni izmed najbolj obiskanih slovenskih spletnih strani (zajeto s programom Ghostery za Google Chrome).

skrbi za dinamično izvajanje vsebine znotraj samega brskalniku [17]. Dinamične spletne aplikacije se v zadnjem obdobju namreč vse bolj izvajajo prav na strani odjemalca zaradi veliko prednosti, ki jih ta nudi. RIA (`Rich Internet Applications`) ali bogate spletne aplikacije so napredne dinamične spletne aplikacije, ki intenzivno uporabljajo datoteke JavaScript za svoje izvajanje. Datoteke takega tipa nas tudi v največji meri zanimajo pri spletnem pajku. Ravno jezik JavaScript namreč omogoča povezovanje med dinamičnimi spletnimi aplikacijami, integracijo zunanjih spletnih aplikacij, nalaganje virov iz zunanjih ponudnikov in izkoriščanje njihovih storitev. S tem se poveča interaktivnost, dinamičnost in prepoznavnost spletnih aplikacij. Med tipičnimi primeri aplikacij, ki jih dinamične spletne aplikacije integrirajo, so vtičniki za storitve podjetja Google, razni vtičniki za družbena omrežja (npr. Facebook, Twitter) in reklamne oglase. Primer integriranih vtičnikov lahko vidimo na Sliki 2.1.

V okviru te diplomske naloge smo tudi mi razvili svojo spletno aplikacijo in sicer kot upravljalno orodje za spletnega pajka. Princip delovanja orodja opišemo v Poglavju 4.

2.2.1 Izvajanje dinamičnih spletnih aplikacij v peskovniku

Dinamične spletne aplikacije največkrat nalagamo s pomočjo spletnih brskalnikov. Spletni brskalniki zahtevajo vire, ki jih aplikacija potrebuje, izvedejo njihovo vsebino in jo ustrezno prikažejo uporabniku. Podobno potrebujemo tudi pri spletnem pajku, vendar z nekaj razlikami. Spletno aplikacijo je potrebno izvesti z mehanizmom peskovnika (ang. `sandbox`) znotraj navideznega spletnega brskalnika. Na tak način je možno spremljati dogodke med nalaganjem aplikacije in znotraj nje izvajati lastno programsko kodo. Pri spremljanju dogodkov med izvajanjem je v našem primeru najbolj pomembno sledenje pridobivanju virov, ki jih aplikacija potrebuje, in shranjevanje vseh njihovih podatkov. Peskovnik nam mora omogočati izvajanje lastne programske kode za namene pridobivanja povezav, ki jih spletna aplikacija vsebuje. Povezave so pomembne za pridobivanje spletnih naslovov naslednjih spletnih aplikacij, ki jih bo spletni pajek izvajal in preiskoval. Orodje, ki smo ga uporabili v ta namen, je opisano v Poglavju 2.3.3.

2.2.2 JavaScript

JavaScript predstavlja izjemno pomemben programski jezik za implementacijo dinamičnih spletnih aplikacij. Pomen tega programskega jezika pri zasnovi in gradnji dinamičnih spletnih aplikacij smo že opredelili v Poglavju 2.2. V diplomskem delu glavno področje interesa pri virih predstavljajo prav datoteke JavaScript, ki se prenašajo za izvajanje spletnih aplikacij. Jezik je pomemben tudi z vidika izdelave rešitve diplomske naloge, saj so bili uporabljeni novejši koncepti, ki jih jezik nudi, in jih omenjamo v naslednjih poglavjih.

JavaScript je lahek (ang. `lightweight`), interpretiran programski jezik, ki nudi podporo objektnemu programiranju. Je dinamično tipiziran jezik, kar pomeni, da spremenljivke ne potrebujejo definicije tipa podatka, ki ga nosijo. Jezik je bil prvotno razvit v podjetju Netscape za namene izvajanja v spletnih brskalnikih. Na ta način spletne strani niso bile več statični HTML, ampak so začele omogočati interakcijo z uporabnikom, nadziranje brskalnika in dinamično spreminjanje ter manipulacijo z elementi jezika HTML [8].

V zadnjih letih je JavaScript izjemno pridobil na popularnosti. Okoli leta 2005 se je začela t. i. AJAX (`Asynchronous JavaScript And XML`) revolucija, ki ga je spremenila v veliko bolj resen programski jezik. S prihodom spletnega brskalnika Google Chrome na tržišče in hudim bojem za uporabnike med brskalniki so se močno razvili in izboljšali pogoni za izvajanje programskega jezika JavaScript. Z bistveno boljšimi pogoni pa se je zgodil preboj še na strežniški strani spletnih aplikacij. Z uporabo Googleovega pogona V8 se je razvilo ogrodje Node.js, ki je strežniško JavaScript okolje s podporo dolgoročnemu izvajanju strežniških procesov [5, 20].

V diplomski nalogi smo poskušali slediti najnovejšim trendom in smo celotno rešitev razvili v programskem jeziku JavaScript. Za implementacijo spletnega pajka smo uporabili izvajalno strežniško okolje, ki je opisano v Poglavju 2.3, za izgradnjo upravljalnega orodja za spletnega pajka pa smo prav tako uporabili novejšo tehnologijo, ki jo razlagamo v Poglavju 4.

2.3 Izvajalno strežniško okolje

Osnovni namen strežnikov je nudenje virov in vsebin s sprejemanjem zahtev, ki jih pošiljajo izvajalniki spletnih aplikacij. Strežniki lahko delujejo tudi kot odjemalci (zahtevajo storitve podatkovnih baz, aplikacijskih strežnikov, drugih spletnih strežnikov). Osnovno izvajalno okolje, ki smo ga uporabili za izvajanje spletnega pajka, je v svoji osnovi strežniško okolje. V našem spletnem pajku to orodje deluje kot strežnik in kot odjemalec. Ena izmed njegovih nalog je zaganjati peskovnik, ki omogoča izvajanje spletnih aplikacij na način, opisan v prejšnjem poglavju (2.2.1). Takrat deluje kot odjemalec. Izvajalno okolje deluje kot strežnik z nudenjem storitev, preko katerih usmerja delovanje peskovnika in celotnega spletnega pajka. Izvajalno okolje je uporabljeno tudi kot spletni strežnik, ki skrbi za nudenje vseh potrebnih vsebin za nemoteno delovanje upravljalnega orodja za spletnega pajka. Podrobneje si bomo delovanje obeh primerov ogledali v Poglavjih 3 in 4.

V Poglavju 2.3.1 je predstavljen koncept, ki se v zadnjem času uveljavlja na področju strežnikov in izpodriva standardne strežniške sisteme. Predstavili bomo izvajalno strežniško okolje Node.js (2.3.2) in orodje PhantomJS (2.3.3), s katerim je omogočeno izvajanje spletnih aplikacij v peskovniku.

2.3.1 Dogodkovno vodeno asinhrono izvajanje

Dogodkovno vodeno asinhrono izvajanje je koncept, ki se v zadnjih letih uveljavlja na področju strežnikov. Gre za strežnike, ki so vodeni dogodkovno, koda pa se izvaja asinhrono z neblokirajočimi V/I (`vhod/izhod`) operacijami. V primerjavi s standardnimi strežniki jim to omogoča elegantnejše rokovanje z več povezavami.

Vsaka nova zahteva predstavlja dogodek. Ob dogodku se izvede povratna funkcija (ang. `callback`). V primeru, da zahtev ni, je strežnik v stanju mirovanja. Asinhrono izvajanje omogoča neblokirajoče V/I operacije. Pri običajnih V/I operacijah, kot je na primer poizvedba v podatkovni bazi, program čaka na rezultat in medtem ne izvaja ničesar. Če v tem času pride do nove zahteve, se to običajno rešuje z delovanjem v več nitih, kar povzroča večjo porabo procesorja in pomnilnika RAM (`Random Access Memory`). Z asinhronim izvajanjem se ne čaka na rezultate V/I operacij, ker se te izvedejo izven glavne programske zanke. To omogoča, da v tem času program ne miruje, ampak nadaljuje z izvajanjem. Posledično to doprinese k manjši porabi sistemskih virov [5].

Na primeru, zapisanem v odseku kode 2.1, lahko vidimo asinhrono izvajanje z neblokirajočo V/I operacijo (poizvedba na podatkovno bazo), katere odgovor je sprožen dogodkovno s povratno funkcijo. Funkcija `newLink.save` izvede shranjevanje objekta v podatkovno bazo. Kot parameter ima podano povratno funckijo, ki se sproži po opravljeni V/I operaciji in sporoči uspešnost shranjevanja, medtem ko se glavni program izvaja naprej. Izpis `Program se izvaja naprej!` bo tako izpisan brez čakanja na rezultat V/I poizvedbe. Izpis v konzolo je torej naslednji:

```
Zacenjam shranjevanje.
Program se izvaja naprej!
Uspesno shranjeno.
```

```
console.log('Zacenjam shranjevanje.');
newLink.save(function(errorDatabaseLinkSave) {
    if (!errorDatabaseLinkSave) console.log('Uspesno shranjeno.');
    else console.log('Napaka!');
});
```

```
console.log('Program se izvaja naprej!');
```

Odsek kode 2.1: Primer dogodkovno vodenega asinhronega izvajanja z neblokirajočo V/I operacijo.

2.3.2 Node.js

Node.js je ogrodje za izgradnjo skalabilnih mrežnih aplikacij v jeziku JavaScript. Zgrajeno je na odprtokodnem pogonu za JavaScript, imenovanem V8, ki deluje tudi znotraj spletnega brskalnika Google Chrome [15]. Node.js omogoča dogodkovno vodeno asinhrono izvajanje, zaradi česar je učinkovito in izjemmno lahko (ang. `lightweight`) ogrodje, ki je zelo prijazno do sistemskih virov.

V diplomski nalogi je ogrodje Node.js uporabljeno kot izvajalno okolje za spletnega pajka. Pri tem sprejema parametre zagona spletnega pajka, zaganja ter nadzira niti, znotraj njega deluje peskovnik, ki izvaja spletne aplikacije, in omogoča povezovanje s podatkovno bazo, v katero shranjuje pridobljene podatke. Deluje tudi kot strežniška stran spletne aplikacije za upravljanje spletnega pajka z nudenjem vseh storitev, ki so potrebne za brezhibno delovanje upravljalnega orodja.

Uporabljena je bila verzija 0.10.29 orodja Node.js za operacijski sistem Windows.

2.3.3 PhantomJS

PhantomJS je orodje, ki smo ga izbrali za izvajanje spletnih aplikacij v peskovniku. Orodje deluje kot običajen spletni brskalnik. Zasnovan je na ogrodju WebKit, ki orodju PhantomJS omogoča izvajanje spletnih aplikacij v peskovniku. Vsebuje JavaScript API (`Application Programming Interface`), preko katerega programiramo in upravljamo njegovo delovanje. Na uradni spletni strani je orodje opisano kot optimalna rešitev za brezglavo (ang. `headless`) testiranje spletnih strani, zajemanje zaslonske slike, dostopanje in manipuliranje spletnih strani preko standarda DOM (`Document Object Model`) in nadziranje omrežja v smislu opazovanja nalaganja spletne strani [12].

WebKit je odprtokodni pogon za spletne brskalnike [14]. Na njem temelji znani spletni brskalnik Safari podjetja Apple. PhantomJS prek JavaScript API-ja nudi

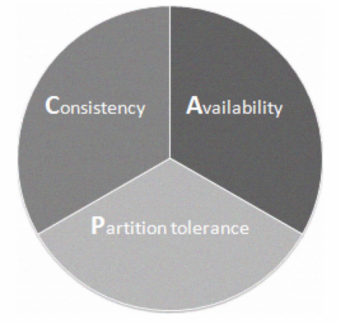
nekaj modulov, ki vsebujejo lepo število metod, parametrov in rokovalnikov. Izvajanje spletnih aplikacij v peskovniku omogoča modul `Web Page`. Preko funkcij modula lahko spremljamo nalaganje spletne aplikacije, izvajamo kodo ob dogodkih, zaznavamo pojavljanje napak in celo izvajamo lastno programsko kodo v spletnih aplikacijah [13].

V okviru diplomske naloge je bil uporabljen PhantomJS verzije 1.9.7 za operacijski sistem Windows.

2.4 Podatkovna baza

Z razvojem interneta in uveljavljanjem računalništva v oblakih se je pojavilo vse več zahtev za večjo zmogljivost podatkovnih baz – predvsem hitrejše branje in pisanje, hranjenje velike količine podatkov (ang. `big data`), skalabilnost ter dostopnost in manjši stroški strojne ter programske opreme. Zahteve so se pojavile predvsem v aplikacijah, ki so široko uporabljene in se zelo hitro širijo. Mednje lahko štejemo storitve družbenih omrežij ali SNS (`Social Networking Service`) in spletne iskalnike. Čeprav so relacijske podatkovne baze precej razširjene, pa kljub temu ne izpolnjujejo omenjenih zahtev po zmogljivosti. Zato so se pojavili novi tipi podatkovnih baz, ki so združene pod skupnim imenom NoSQL. Omogočajo boljše zmogljivosti pri skalabilnosti in dostopnosti in so zasnovane za hranjenje velikih količin podatkov [10].

Večinoma so baze NoSQL porazdeljene med več strežniki, kar jim omogoča veliko V/I operacij na sekundo. Glavna razlika v primerjavi z relacijskimi PB (`podatkovna baza`) je v tem, da baze NoSQL praviloma ne podpirajo jezika SQL (`Structured Query Language`) in z njim ACID (`Atomicity, Consistency, Isolation and Durability`) transakcij, ker jih največkrat ne potrebujejo. Spremembe v bazi se ponavadi šele sčasoma razširijo po celem sistemu. Kot nasprotje principu ACID se uveljvalja princip BASE (`Basically Available, Soft state, Eventually consistent`), ki pomeni, da so podatki v osnovi razpoložljivi in šele sčasoma konsistentni. Z opustitvijo principa ACID so PB veliko bolj zmogljive in skalabilne v zameno za počasnejše posodabljanje sistema in morebitno neenakostjo ali netočnostjo vseh podatkov, ki pa niso kritičnega pomena, če se jih aplikacija zaveda. Kljub vsemu



Slika 2.2: Teorem CAP (vir: [10]).

obstajajo mehanizmi, ki lahko zagotovijo neko mero konsistentnosti. Pojavile so se tudi nove relacijske PB, ki omogočajo bistveno večjo skalabilnost v primerjavi z osnovnimi relacijskimi PB, a hkrati ohranjajo princip ACID in poizvedbe v jeziku SQL [6, 10].

V povezavi s podatkovnimi bazami NoSQL se velikokrat omenja teorem CAP (`Consistency, Availability, Partition tolerance`) Erica Brewerja (Slika 2.2). Kratica predstavlja konsistenco (ang. `consistency`), razpoložljivost (ang. `availability`) in toleranco do particioniranja (ang. `partition tolerance`), teorem pa pravi, da porazdeljen sistem nikoli ne more ustreči vsem trem zahtevam, temveč kvečjemu dvema. Večina PB NoSQL se zato odreče konsistenci, ni pa to nujno [6, 10].

Podatkovne baze NoSQL se med seboj ločujejo po načinu shranjevanja podatkov. V vseh načinih podatke shranjujejo v parih atribut–vrednost, uporabljajo pa tri različne podatkovne strukture [6], in sicer podatkovno strukturo "ključ–vrednost", "dokument" in stolpčasto orientirano podatkovno strukturo.

Podatkovna struktura "ključ–vrednost" (ang. `key-value`) uvaja en indeks (ključ) za vse vnose, ki ga določi programer. Kot vrednost so shranjeni ostali atributi vnosa, ki ga unikatno določa ključ. Primer podatkovne baze, ki uporablja tako strukturo, je Redis. Strežniki, ki uporabljajo Redis, shranjujejo podatke v pomnilnik RAM, kar omogoča izredno hitro branje in pisanje. Podatki se lahko asinhrono prenašajo na disk za namene varnostnega kopiranja. Poleg nizov in zapisov BLOB (`Binary`

Large Object) podpira tudi sezname in množice kot ključ, ki se indeksira. Podatkovna baza je primerna za sisteme, ki potrebujejo hitre računske operacije, ni pa primerna za velike količine podatkov [6, 10].

Podatkovna struktura "dokument" je zelo podobna strukturi "ključ–vrednost" z razliko, da so vrednosti shranjene v formatu JSON (JavaScript Object Notation) ali XML (ang. Extensible Markup Language), kar omogoča bolj kompleksno strukturo podatkov. Sistemi ponavadi dopuščajo tudi sekundarne indekse in več različnih struktur dokumentov (objektov) znotraj iste zbirke v PB [6]. Če to prevedemo v izrazoslovje relacijskih PB, je to enako, kot če bi te dopuščale vnose s poljubnimi atributi za vsak vnos znotraj iste tabele. Kot primer PB NoSQL s tako podatkovno strukturo je v Poglavju 2.4.1 opisana PB MongoDB, ki je bila tudi izbrana za realizacijo diplomske naloge.

Osnovni podatkovni model stolpčasto orientirane (ang. column-oriented) podatkovne strukture so vrstice in stolpci, ki so porazdeljeni med številna vozlišča sistema. Vrstice so razdeljene glede na drobljenje primarnega ključa, stolpci pa glede na grupiranja atributov. Oba načina particioniranja (vertikalno in horizontalno) sta lahko uporabljena hkrati na isti tabeli. Vrstice imajo lahko različno število atributov. Skupine atributov za grupiranje pa morajo biti vnaprej definirane. Podatkovna struktura temelji na Googleovem BigTableu, ki ga uporablja veliko število Googleovih storitev. Hrani velike količine podatkov (več petabajtov) prek več tisoč strežnikov. Uporablja stiskanje podatkov in izredno zmogljivo paralelno procesiranje. Aplikacije imajo zelo različne zahteve glede velikosti podatkov (od URL-jev do satelitskih posnetkov) in latence (ang. latency), sistem pa mora ustreči vsem. Podobno podatkovno strukturo uporablja tudi Facebook s Cassandro in Yahoo s PNUTSem [6, 7, 10].

2.4.1 MongoDB

V diplomski nalogi smo se odločili za delo s PB NoSQL. Pri preiskovanju spletnih aplikacij smo želeli vse pomembne pridobljene podatke hraniti. Glede na obsežnost svetovnega spleta je pričakovati, da bo preiskanih spletnih aplikacij ogromno, še več pa bo na razpolago podatkov o njih. Prav tako si želimo čim hitrejše shranjevanje

SQL izrazi	MongoDB izrazi
podatkovna baza (ang. `database`)	podatkovna baza (ang. `database`)
tabela (ang. `table`)	zbirka (ang. `collection`)
vrstica (ang. `row`)	dokument (ang. `document`)
stolpec (ang. `column`)	polje (ang. `field`)

Tabela 2.1: Izrazi za komponente relacijskih podatkovnih baz v primerjavi z MongoDB NoSQL podatkovnimi bazami.

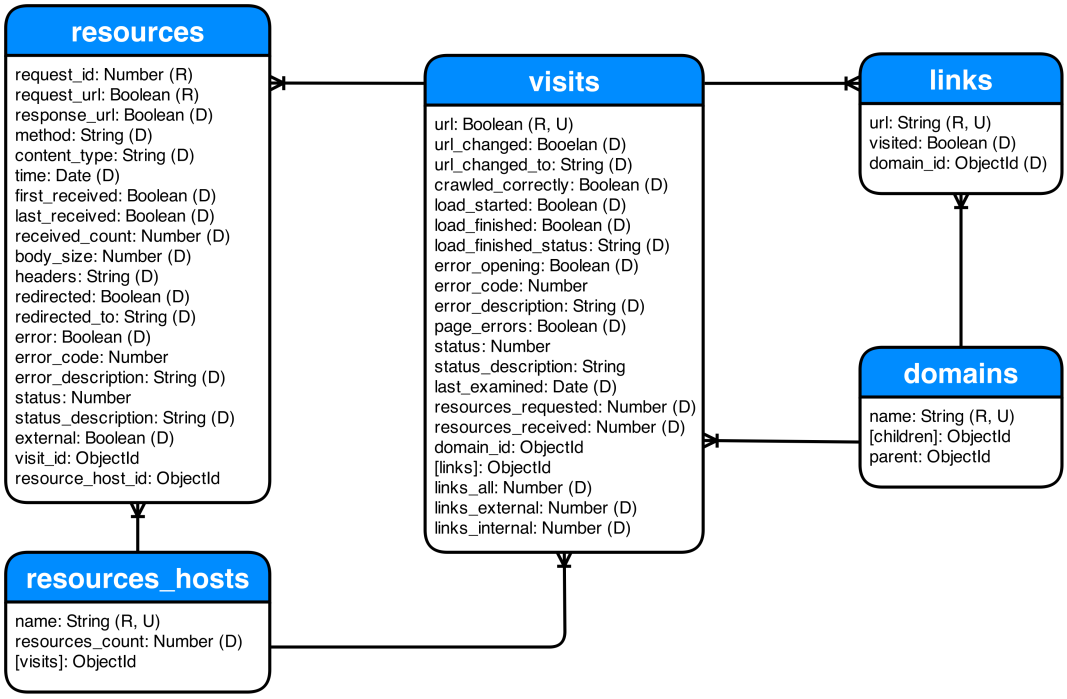
v bazo. Vse omenjene lastnosti naredijo PB NoSQL izrazito bolj primerno izbiro kot relacijsko PB. Po pregledu PB NoSQL smo se odločili za dokumentno podatkovno strukturo zaradi možnosti sekundarnih indeksov in shranjevanja v formatu JSON, znotraj nje pa smo izbrali MongoDB. MongoDB smo izbrali zaradi pozitivnih izkušenj in predvsem zaradi lupine (ang. `shell`), ki nudi JavaScript izvajalno okolje. To se lepo dopolnjuje z našo željo, da celotna naša rešitev v čim večji meri temelji na tem programskem jeziku. Poleg vsega tega je MongoDB eden najbolj znanih in razširjenih ponudnikov PB NoSQL.

MongoDB je torej PB NoSQL z dokumentno podatkovno strukturo. Dokumenti se shranjujejo v formatu BSON (`Binary JSON`), ki je binarno kodirana oblika formata JSON. V primerjavi z relacijskimi PB, MongoDB uvaja nekoliko drugačne izraze za komponente PB, ki so vidni v Tabeli 2.1. Nudi osnovne CRUD (`Create, Read, Update, Delete`) operacije kot tudi zahtevnejše agregacije. Omogoča tudi indeksiranje poljubnih atributov oziroma polj [16].

V diplomski nalogi je bil uporabljen MongoDB verzije 2.6.3 za sisteme Windows.

2.4.2 Zasnova podatkovnega modela

Preden se je začel razvoj spletnega pajka, je bilo potrebno določiti podatkovni model. Shema podatkovnega modela je vidna na Sliki 2.3. V osnovi je bil določen glede na načrtovan cilj uporabe spletnega pajka, dodatno pa razširjen glede na podatke, ki jih je bilo moč pridobiti z uporabljenimi orodji. V ta namen so bili določeni naslednji entitetni tipi:



Slika 2.3: Shema podatkovnega modela (R – `required`, U – `unique`, D – `default`).

- `visits` – predstavlja obiskano spletno aplikacijo z določenim naslovom URL in osnovnimi podatki o njenem nalaganju. Vsebuje polje vseh referenc na najdene povezave in referenco na domeno.

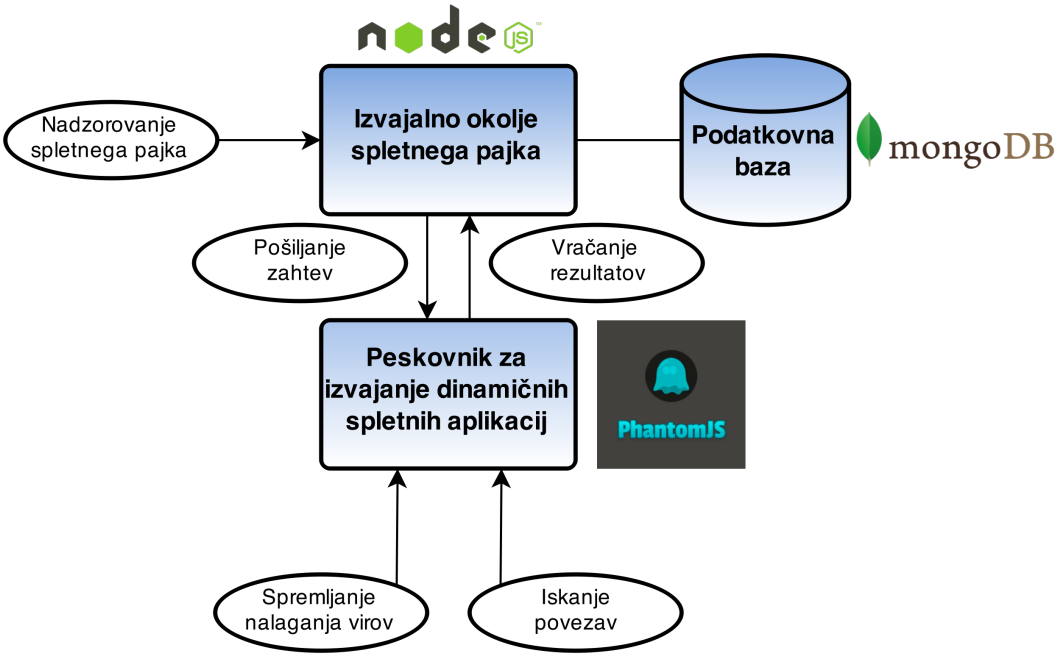
- `links` – predstavlja vse najdene povezave na spletnih aplikacijah in hkrati predstavlja vir, od koder spletni pajek jemlje URL-je za naslednje obiske spletnih aplikacij. Lahko vsebuje referenco na domeno.

- `resources` – predstavlja vse naložene vire. Vsebuje referenco na spletno aplikacijo (obisk), na katerega je bil prenesen, ter referenco na gostitelja virov s katerega je bil prenesen.

- `resources_hosts` – predstavlja gostitelje virov, s katerih se prenašajo viri na spletne aplikacije. Vsebuje referenco na obiske, ki so nalagali vire iz gostitelja virov.

- `domains` – predstavlja hierarhijo domen. Hierarhije se gradi tako, da se naslov `http://www.domena.si` razbije na entitete `si`, `domena.si` in `www .domena.si`, pri čemer se med seboj referencirajo. Vsaka entiteta vsebuje starša in seznam otrok. V danem primeru ima entiteta `si` kot otroka entiteto `domena.si`, starša nima. Entiteta `domena.si` ima otroka `www.domena.si`, starš je entiteta `si`. Entiteta `www.domena.si` nima otrok, kot starša pa ima entiteto `domena.si`.

Atribute entitet smo določili glede na potrebne reference, podatke, ki se jih je dalo zajeti s spletnim pajkom, in glede na napake, ki se lahko pojavijo med zajemanjem in shranjevanjem podatkov. Vse entitete vsebujejo unikatni identifikator `_id`, ki ga avtomatično generira MongoDB, pri čemer gre za 12-bajtni BSON [16]. Privzete vrednosti so pri vseh tipih spremenljivke `Boolean` nastavljeni na false, tipi `Date` so nastavljeni na trenutni datum in uro, atributi tipa `Number` so nastavljeni na 0 (izjema je pri gostiteljih virov, kjer je atribut `resources_count` privzeto nastavljen na 1), atributi tipa `String` in `ObjectId` pa so nastavljeni na `null`. Seveda to velja le za atribute, kjer je privzeta vrednost določena. V shemi na Sliki 2.3 so prikazane omejitve, kjer črka R predstavlja obvezno polje (ang. `required`), črka U predstavlja unikatno polje (ang. `unique`) v zbirki, črka D pa pomeni, da ima polje



Slika 2.4: Shema arhitekture in orodij spletnega pajka.

določeno privzeto (ang. `default`) vrednost. Vse enitete imajo poleg atributov, napisanih v shemi, še 3 atribute. Prvi je že omenjeni `_id`, drugi in tretji pa sta atributa `created_at` in `changed_at`, uporabljena pa sta kot časovni oznaki (ang. `timestamp`) za čas, ko se je dokument ustvaril oziroma nazadnje spremenil v PB.

2.5 Arhitektura predlagane rešitve

Arhitektura našega spletnega pajka je prikazana na Sliki 2.4. Osnova spletnega pajka je torej izvajalno okolje, za kar je v našem primeru uporabljen Node.js. S konceptom dogodkovno vodenega asinhronega izvajanja, ki ga omogoča, si želimo, da bi bil pajek čim bolj prijazen sistemskim virom in hkrati čim hitrejši ter dobro izkoriščen pri svojem delovanju. Izvajalno okolje nadzira in usmerja pajkovo delovanje. Omogoča tudi povezavo s podatkovno bazo NoSQL, za kar smo izbrali MongoDB. Pričakujemo veliko število podatkov, ki se bodo shranjevali konstantno

med sprehajanjem, zato tudi izbira PB NoSQL. Pajka sestavlja še najpomembnejši del – peskovnik za izvajanje spletnih aplikacij. Ta omogoča nalaganje aplikacij, ki jih želimo preiskati. S svojimi orodji omogoča sledenje virom, ki nas zanimajo. Omogoča izvajanje lastne programske kode znotraj aplikacije, s čimer želimo pridobiti povezave, ki jih spletna aplikacija vsebuje. Za peskovnik smo uporabili orodje PhantomJS, ki prav tako podpira dogodkovno vodeno asinhrono izvajanje. Izvajalno okolje nadzira delovanje peskovnika in shranjuje podatke, ki jih ta pridobi, v PB.

Poglavje 3

Razvoj spletnega pajka

V Poglavju 3 je opisan razvoj naše implementacije spletnega pajka. V podpoglavjih je predstavljen način povezave izvajalnega okolja s podatkovno bazo, kako je v izvajalno okolje integrirano orodje, ki omogoča rabo peskovnika (3.2), iz katerih programskih komponent je pajek sestavljen (3.3), v zadnjem podpoglavju pa je opisano podrobnejše delovanje našega spletnega pajka (3.4).

3.1 Povezava izvajalnega okolja s podatkovno bazo

Pri povezavi izvajalnega okolja Node.js s podatkovno bazo smo uporabili modul `mongoose`, ki omogoča povezavo na PB MongoDB. Za definicijo podatkovnih modelov omogoča sistematičen pristop z uporabo shem in modelov. Ponuja pretvorbo tipov podatkov (ang. `casting`), validacijo, omejitve in poizvedbe v PB s podporo dogodkovno vodenim asinhronim izvajanjem z neblokirajočimi V/I operacijami (Poglavje 2.3.1). Poizvedbe so tako izvedene na način, ki najbolj sovpada z delovanjem izvajalnega okolja Node.js.

Izdelava shem in modulov je preprosta, rezultat pa je objekt v JavaScriptu, ki predstavlja dokument v PB. Primer izdelave sheme za entitetni tip domene je viden v Odseku kode 3.1. Najprej se določi shema z vsemi atributi, tipi podatkov in omejitvami. S funkcijo `mongoose.model` se generira zbirka `domains` v PB, vrnjeni

razred pa je pripravljen za uporabo. Primer poizvedbe shranjevanja v PB je že predstavljen v Odseku kode 2.1, kjer smo predstavili dogodkovno vodeno asinhrono izvajanje, vse ostale operacije pa se izvajajo na zelo podoben način.

```
var domainSchema = mongoose.Schema({
    name: { type: String, required: true, unique: true },
    children: [{ type: ObjectId, ref: 'domain' }],
    parent: { type: ObjectId, ref: 'domain' },
    created_at: { type: Date, default: Date.now },
    changed_at: { type: Date, default: Date.now }
});

var Domain = mongoose.model('domain', domainSchema);
```

Odsek kode 3.1: Primer izdelave sheme v modulu `mongoose` za entitetni tip domene.

3.2 Integracija izvajalnega okolja s peskovnikom za izvajanje spletnih aplikacij

V prvi vrsti je bilo glede na izbrano zasnovo spletnega pajka potrebno ugotoviti, ali bomo uspeli integrirati orodje PhantomJS, ki deluje kot izvajalec spletnih aplikacij, z izvajalnim okoljem Node.js. PhantomJS kot povsem samostojna aplikacija sicer omogoča enostavno in efektivno nalaganje ter manipulacijo s spletnimi stranmi, vendar pa kot samostojno orodje ni dovolj zanesljivo in močno za realizacijo spletnega pajka. V začetku smo namreč poskušali izvesti celotnega spletnega pajka le z orodjem PhantomJS, vendar smo hitro spoznali, da to ne bo izvedljivo. Takrat smo začeli razmišljati o orodju Node.js kot izvajalnem orodju, kar se je na koncu izkazalo za veliko boljšo rešitev.

Na spletu je možno zaslediti kar nekaj nasvetov v zvezi z integracijo omenjenih orodij. Odkrili smo nekaj modulov, ki to omogočajo. Po pregledu in preizkusu delovanja kar nekaj variant smo izbor zožili na dva modula, in sicer `node-phantom-simple` in `node-phantom`. Modula sta si zelo podobna, saj je `node-phantom-simple` nastal iz modula `node-phantom`. `Node-phantom-simple` ima poenostavljen način komunikacije z orodjem Node.js, uporablja manj odvisnosti in manj plasti delovanja

[18]. Pri testiranju obeh smo se na koncu odločili za modul `node-phantom-simple`, saj je deloval hitreje, obenem pa se je izkazal za nekoliko bolj stabilnega.

Samo nadzorovanje orodja PhantomJS preko modula je enostavno, uporaba pa skoraj identična kot v JS API-ju. Razlika je le v določanju parametrov posameznih modulov orodja, do katerih prek ogrodja Node.js ni možno dostopati neposredno, ampak prek metod `get` in `set`. Nekoliko drugačno je tudi podajanje parametrov za zagon orodja PhantomJS, saj to ni več zaganjano direktno iz konzole, temveč prek funkcije modula. Parametri zagona so zato podani kot parameter funkcije.

Potrebno je tudi omeniti, da smo pri integraciji orodij naleteli na počasno nalaganje določenih spletnih aplikacij, ki so občasno celo povzročale sesutje orodja PhantomJS. Problem smo našli zabeležen na številnih spletnih straneh, kot rešitev pa je bil predlagan zagon s parametrom `proxy-type: 'none'`, ki onemogoči uporabo posredovalnega strežnika (ang. `proxy server`). Težava se sicer pojavlja zgolj na sistemih Windows. Z omenjeno rešitvijo smo v veliki meri odpravili težave.

Sam modul `node-phantom-simple` smo nekoliko preuredili tako, da je ob zagonu v funkcijo sprejel še dodatna parametra – identifikator niti in povratno metodo za primere napak. S prvo smo omogočili sledljivost več nitim oziroma instancam orodja PhantomJS, ki smo jih uporabljali za hitrejšege sprehajanje po spletnih aplikacijah, druga pa je služila za zaznavanje in rokovanje z nepričakovanimi ustavitvami orodja PhantomJS.

V Odseku kode 3.2 je prikazan način zagona orodja PhantomJS prek izbranega modula v izvajalnem okolju Node.js brez uporabe posredovalnega strežnika. Prvi podan parameter funkcije `nodePhantomSimple.create`, ki požene orodje, podaja povratno funkcijo ob zagonu orodja, drugi parameter podaja parametre zagona orodja PhantomJS, tretji podaja identifikator niti in zadnji podaja povratno funkcijo za primere napak.

```
var nodePhantomSimple = require('node-phantom-simple');
nodePhantomSimple.create(
    onPhantomStartedCallback,
    { parameters: { proxy-type: 'none' } },
    threadId,
    onPhantomErrorCallback
```

```
);
```

Odsek kode 3.2: Zagon orodja PhantomJS znotraj izvajalnega okolja Node.js.

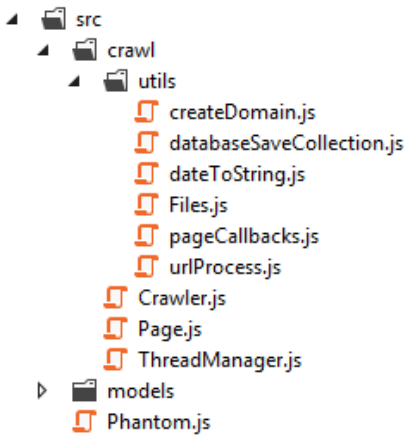
3.3 Komponente spletnega pajka

Koda spletnega pajka je razdeljena v več komponent glede na funkcionalnosti, ki jih opravljajo. Na Sliki 3.1 je vidna struktura datotek oziroma komponent izvorne kode spletnega pajka. V osnovi je spletni pajek sestavljen iz štirih razredov, ki opravljajo različne naloge:

- `Phantom` – deluje kot ovojni razred za celotnega pajka. Znotraj njega se odpre in zapre povezava na PB. Zadolžen je za izpis končnega poročila o sprehajanju.
- `ThreadManager` – kot že samo ime pove, se tu nahajajo funkcije za ravnanje in rokovanje z nitmi. Zaganja nove niti (kot nove objekte) in nadzoruje njihovo delovanje, skrbi za izpis poteka in štetje preiskanih strani. Ob zagonu spletnega pajka je zadolžen za enakomerno razdelitev semenskih spletni naslovov URL med niti.
- `Crawler` – znotraj tega razreda se zažene instanca orodja PhantomJS. Objekt skrbi za nadzor njegovega delovanja, zaznavanja napak, pridobivanja novega URL-ja za preiskovanje, ob izbranem URL-ju je zadolžen za generiranje hierarhije domen in za ustrezno ravnanje ob preiskani spletni aplikaciji.
- `Page` – predstavlja spletno aplikacijo, ki jo izvaja PhantomJS. Znotraj se izvede nalaganje spletne aplikacije, zagon vseh povratnih funkcij orodja PhantomJS in iskanje povezav v spletni aplikaciji. Ob koncu zažene shranjevanje vseh pridobljenih podatkov v PB.

Poleg omenjenih razredov spletni pajek uporablja še nekaj orodij:

- `createDomain` – orodje, ki poskrbi za kreiranje hierarhije domen iz danega naslova URL. Deluje od zgoraj navzdol – najprej procesira najvišjo domeno, potem pa se pomika navzdol. V prvem koraku preveri, če entiteta za nivo



Slika 3.1: Struktura izvorne kode spletnega pajka.

domene že obstaja. Če ne obstaja, jo ustvari v bazi. Poiskati mora tudi starša, ki mu posodobi otroke, ustvarjeni entiteti pa določi najdenega starša.

- `databaseSaveCollection` – skrbi za shranjevanje večjih količin podatkov hkrati. Uporabi se po preiskani spletni aplikaciji in se uporablja za shranjevanje povezav, virov in gostiteljev virov. Šteje vse uspešno shranjene entitete in doda njihov izpis v poročilo preiskane spletne aplikacije.

- `dateToString` – nudi več funkcij z različnimi formati izpisa in pretvorb datumov v niz za potrebe spletnega pajka.

- `Files` – razred, ki ustvari datoteke za beleženje in omogoča pisanje vanje. Beleženje poteka za vsako nit posebej, in sicer se beleži shranjevanje v PB znotraj preiskovane spletne aplikacije. Izpisuje se status shranjevanja notranjih ter zunanjih povezav in notranjih ter zunanjih virov. Izpisi so bolj namenjeni podrobnemu pregledovanju shranjevanja in pri iskanju napak v delovanju.

- `pageCallbacks` – skrbi za povezovanje povratnih funkcij z rokovalniki orodja PhantomJS in njihovo izvajanje.

- `urlProcess` – procesira podani URL in poskuša izločiti neveljavne, še preden se shranijo v PB. Sestavlja URL-je, ki referencirajo znotraj lokalne spletne

aplikacije in ne uporabljajo absolutnega naslova, ampak relativno pot glede na trenutni URL.

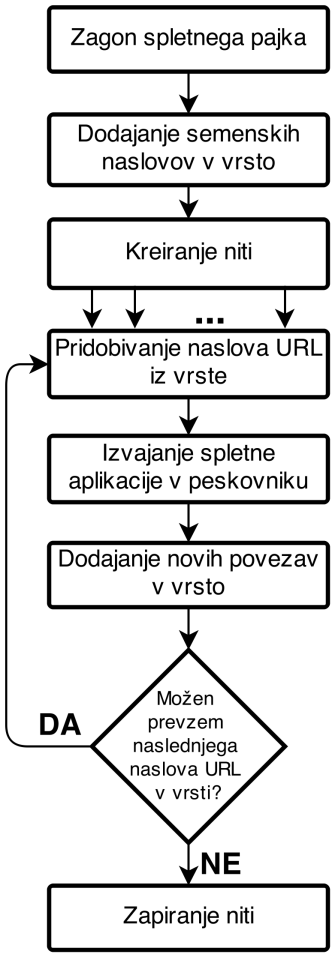
3.4 Delovanje spletnega pajka

V tem poglavju je po korakih razloženo delovanje naše implementacije spletnega pajka od njegovega zagona do izvajanja spletne aplikacije v peskovniku, pridobivanja podatkov in shranjevanja vseh podatkov v PB. Shema diagrama poteka delovanja spletnega pajka je vidna na Sliki 3.2.

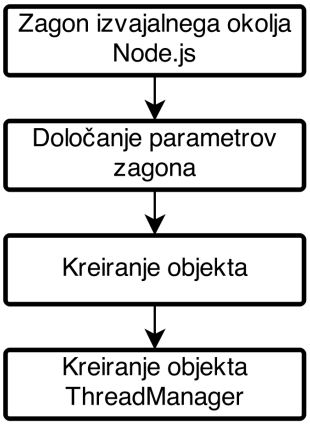
3.4.1 Zagon spletnega pajka

Za zagon spletnega pajka je potrebno izvajalno okolje Node.js in orodje PhantomJS. Shema diagrama zaganjanja je prikazana na Sliki 3.3. Spletnega pajka zaženemo tako, da kreiramo nov objekt `Phantom`, ki mu podamo zahtevane parametre, ki določajo delovanje. Ti parametri so:

- `initialUrls` – seznam semenskih naslovov URL. Te bo spletni pajek preiskal najprej. V primeru, da je podan prazen seznam, bo pajek poskusil pridobiti naslove iz PB. Pridobivanje je podrobneje opisano v Poglavju 3.4.3.
- `numPagesToCrawl` – število spletnih naslovov, ki naj jih spletni pajek preišče. Dejansko število preiskanih naslovov lahko variira zaradi napak, ki se lahko zgodijo med sprehajanjem. Parameter deluje kot omejitev sprehajanja, da se to ne izvaja v nedogled.
- `numThreads` – število niti oziroma število paralelno delujočih instanc orodja PhantomJS. Možnost izbire je dana zaradi različnih zmogljivosti sistemov, na katerih je bil pajek testiran. Število niti, s katerimi lahko optimalno operira spletni pajek, je namreč odvisno od zmogljivosti procesorja in količine pomnilnika RAM, ki nam je na voljo.
- `waitTime` – število milisekund, ki jih spletni pajek počaka, da se spletna aplikacija naloži, preden začne z obdelavo podatkov.



Slika 3.2: Shema diagrama poteka delovanja spletnega pajka.

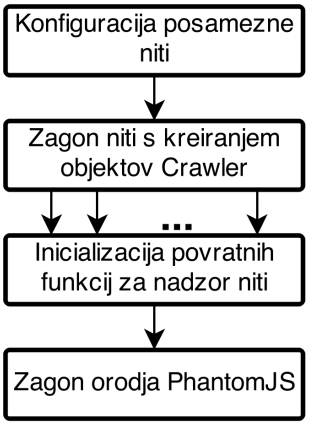


Slika 3.3: Shema diagrama poteka zaganjanja spletnega pajka.

- `dbConnectionString` – naslov do podatkovne baze MongoDB.
- `phantomJSPath` – pot do izvajalne datoteke orodja PhantomJS. Če je vrednost parametra `null`, se bo orodje naložilo iz okoljske spremenljivke `PATH`.
- `labelLinksDomain` – vrednost je lahko `true` ali `false`. Pajek omogoča gradnjo hierarhije domen tudi iz vseh najdenih povezav in ne le iz spletnega naslova naloženih spletnih aplikacij.
- `maxDepth` – maksimalna globina naslova URL, ki jo merimo na sledeči način: `http://globina.url.si/1/2/3`.
- `blacklist` – seznam domen, za katere ne želimo, da jih pajek preiskuje. Domene bodo vseeno pristale v hierarhiji v primeru, da pajek do njih pride, spletne aplikacje pa se ne bodo izvajale.

Po ustvarjenem objektu `Phantom` s podanimi parametri se vzpostavi povezava na PB. Ob uspešno vzpostavljeni povezavi se zažene `ThreadManager` oziroma upravljalec niti.

Za diplomsko nalogo smo za zagon spletnega pajka v času razvoja uporabljali preprosto skripto, ki se je zagnala iz konzole in je s podanimi argumenti kreirala parametre za zagon. Kasneje se je ustvarila še ena skripta za zagon pajka iz upravljalnega orodja, več o tem pa v Poglavju 4.3.2.



Slika 3.4: Shema diagrama poteka kreiranja posameznih niti spletnega pajka.

3.4.2 Ustvarjanje niti in nadzor nad njimi

Shema diagrama poteka zaganjanja niti je vidna na Sliki 3.4. Prva naloga upravljalca niti je razdelitev začetnih semenskih naslovov mednje. Zasnovan je tako, da se semenski naslovi porazdelijo čim bolj enakomerno v začetne vrste spletnih naslovov posameznih niti. Upravljalec mora niti ustrezno skonfigurirati glede na parametre delovanja spletnega pajka. Takoj zatem začne z zaganjanjem niti prek kreiranja objektov `Crawler`, ki skrbijo za peskovnike prek več instanc orodij PhantomJS. Vsak objekt s pripradajočo instanco orodja PhantomJS predstavlja eno nit spletnega pajka. Upravljalec nastavi tudi nekaj števcev, prek katerih beleži in nadzoruje delovanje niti. Upravljalec niti oštevilči, in sicer od števila 0 naraščajoče.

Nadzorovanje niti je razdeljeno med razreda `ThreadManager` in `Crawler`. Kot je že bilo omenjeno, je kar nekaj težav povzročala nestabilnost orodja PhantomJS. Prav te napake se rešujejo znotraj razreda `Crawler`, saj se tam nahaja povratna metoda, ki orodje ponovno zažene. To je mogoče, ker so izvajanja orodja PhantomJS v vsaki niti neodvisna med seboj. Razred `Crawler` skrbi tudi za ponovno zaganjanje orodja na vsakih 20 uspešno preiskanih spletnih naslovov. Za ta ukrep smo se odločili zaradi nepojasnjenega stalnega povečevanja porabe pomnilnika (ang. `memory leak`), ki so ga zaznali tudi številni drugi uporabniki orodja PhantomJS. Ukrep je zelo preprost, a učinkovit, ponovno zaganjanje pa ni tako zamudno, da bi lahko bistveno vplivalo

```
------------ CRAWLING PAGE #2 (THREAD #5) -------------

10:10:41 - Opening page (http://www.vlada.si/koristno/).
10:10:42 - Url changed to http://www.vlada.si/koristno/ostalo/404_error_page_not_found/
10:10:43 - Started evaluating page.
10:10:43 - Found 56 urls.
10:10:43 - Adding 20 urls.
10:10:43 - End of evaluation.
10:10:44 - Page exiting.

Page scanned in 3s 499ms.
0 links successfully saved to database.
22 resources successfully saved to database.
Resources received: 22/23 (95.7%)


THREAD MANAGER REPORT
|=                                    | (9/500, 1.80%)
```

Slika 3.5: Izgled izpisa poteka sprehajanja v konzoli.

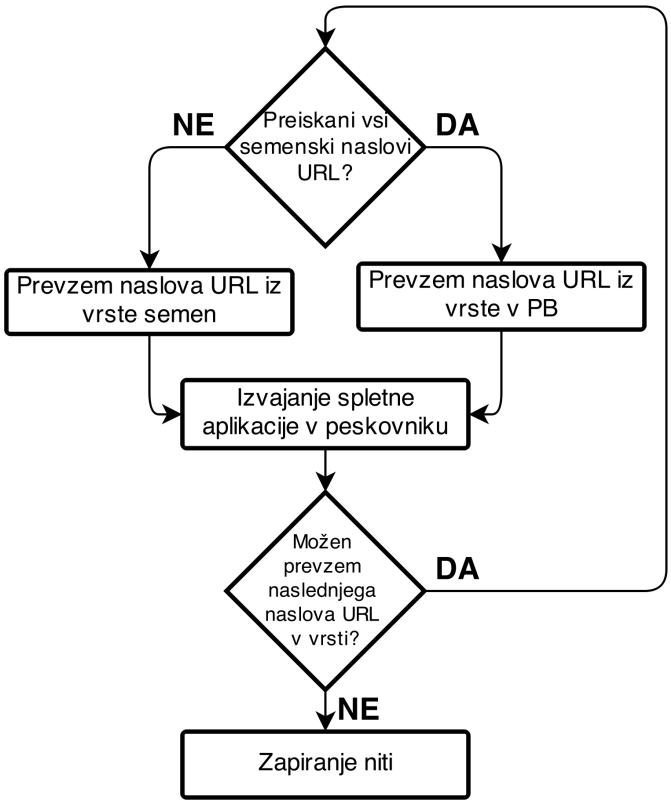
na zmogljivost pajka.

Upravljalec niti nudi še nekaj podpornih funkcij za usmerjanje delovanja. Prva funkcija se uporabi po uspešno preiskani spletni aplikaciji. Upravljalec osveži svoje števce in poveča število preiskanih naslovov, hkrati pa niti sporoči, ali naj nadaljuje s prevzemom naslednjega naslova ali pa je preiskanih dovolj naslovov in naj zaključi z delovanjem. Druga funkcija zgolj glede na identifikator niti sporoči, ali je ta aktivna ali ne. Ta funkcija se rabi pri ponovnem zagonu orodja PhantomJS, saj se lahko ta sesuje ob zapiranju in je nit že zaprta na strani upravljalca, samo orodje pa bo še vedno poskušalo izvesti ponoven zagon. Deluje torej kot nekakšno varovalo pred nedovoljenim ponovnim zagonom. Tretja funkcija služi zapiranju niti. Ko te zaključijo s svojim delovanjem, mora upravljalec nit ustrezno zapreti in to označiti v svojih spremenljivkah.

Upravljalec skrbi tudi za izpis poteka sprehajanja (Slika 3.5). Ob vsaki uspešno preiskani spletni aplikaciji izpiše število in odstotek preiskanih aplikacij zaradi lažjega sledenja ter vidnosti napredka sprehajanja.

3.4.3 Pridobivanje naslova URL

Shema diagrama poteka pridobivanja naslova URL je vidna na Sliki 3.6. Pridobivanje naslova URL poteka s prevzemanjem iz vrst tipa FIFO (`First In, First Out`). To pomeni, da bodo najprej preiskai spletni naslovi, ki so bili prvi dodani v vrsto.



Slika 3.6: Shema diagrama poteka pridobivanja naslova URL.

Obstajata dve vrsti, in sicer vrsta v PB in vrsta, ki vsebuje semenske naslove URL podane kot parameter ob zagonu spletnega pajka. Vedno ima vrsta s semenskimi naslovi prednost pred vrsto v PB. Torej šele ko so preiskani vsi semenski naslovi, ki jih je nit sprejela, se začne prevzemati naslove URL iz podatkovne baze.

Da pride do prevzemanja spletnih naslovov, je potrebnih še nekaj preverjanj. V primeru, da je bila nit ravno ustvarjena ali pa PhantomJS ponovno zagnan iz kakršnegakoli razloga, se nit pomakne direktno v prevzem naslova. To se zgodi tudi takrat, ko se znotraj niti zgodi napaka pri izvajanju spletne aplikacije ali shranjevanju podatkov oziroma vsakič, ko se pregled spletne aplikacije konča z neuspešno kodo. V primeru uspešno preiskane aplikacije pa je potrebno najprej pri upravljalcu niti preveriti, če sploh lahko nadaljujemo s sprehajanjem. Upravljalec to dovoli v

primeru, da še ni bilo preiskanih dovolj spletnih aplikacij. V nasprotnem primeru se mora nit zapreti.

Pred prevzemom naslova URL se tudi izvede ponoven zagon orodja PhantomJS, v kolikor je to potrebno. To se zgodi na vsakih 20 uspešno preiskanih aplikacij.

Pri prevzemu spletnega naslova iz PB mora nit izvesti poizvedbo. Naslovi se hranijo v zbirki povezav, z atributom `visited` pa je označeno, ali je bila povezava že prevzeta. Pajek torej naredi poizvedbo v PB po neobiskanih povezavah, kjer vzame prvi rezultat, rezultati pa so razporejeni v vrstnem redu dodajanja, s čimer dobimo vrsto FIFO.

Ko ima pajek prevzet naslov URL za naslednji pregled spletne aplikacije, preveri, če je zanj že zgenerirana hierarhija domen (z iskanjem celotne domene v PB). V primeru da je ni, se aktivira generiranje hierarhij preko orodja `createDomain`. Ta operacija je nekoliko bolj zahtevna, saj lahko zahteva kar nekaj poizvedb, shranjevanj in posodobitev v PB, ki morajo biti zaradi odvisnosti podatkov nujno izvedene zaporedno in ne morejo polno izkoristiti dogodkovno vodenega asinhronega izvajanja.

Ob pridobljenem naslovu in generirani domeni se lahko prične izvajanje spletne aplikacije v peskovniku.

3.4.4 Pridobivanje in obdelava podatkov

Shema diagrama poteka izvajanja dinamične spletne aplikacije v peskovniku je vidna na Sliki 3.7. Izvajanje spletne aplikacije se izvede znotraj objekta `Page`, ki se mu poda naslov URL in generirano domeno. Pajek najprej ustvari novo entiteto v zbirki obiskov z danim naslovom. Če pri shranjevanju pride do podvojenega vnosa (obiski imajo v PB določeno omejitev, pri kateri morajo biti naslovi URL unikatni), pomeni, da je stran že bila obiskana. Objekt javi neuspešno preiskano spletno aplikacijo. Ob uspešno shranjeni entiteti pa se lahko začne izvajanje dinamične spletne aplikacije.

Izvajanje dinamične spletne aplikacije v peskovniku

Spletna aplikacija se naloži s klicem funkcije `createPage` znotraj orodja PhantomJS. Povratna funkcija nam vrne objekt `page`, ki predstavlja peskovnik, prek ka-

Slika 3.7: Shema diagrama poteka izvajanja dinamične spletne aplikacije v peskovniku.

terega nadziramo izvajanje aplikacije. V tem objektu se najprej incializira nekaj rokovalnikov, med drugim tudi za pridobivanje podatkov o virih, zaznavanje napak ipd. Zatem lahko odpremo spletno aplikacijo s funkcijo `open` objekta `page`, ki ji podamo izbran naslov URL. Pri klicu funkcije smo v nastavitvah določili, da pajek ne nalaga slik. Za ta korak smo odločili zaradi težav, ki jih slike občasno povzročajo orodju PhantomJS. Prav tako nas nalaganje slik v tej diplomski nalogi ne zanima. Po odpiranju pajek počaka število milisekund, ki je določeno v parametru `waitTime`, podanem ob zagonu spletnega pajka, preden se začnejo iskati povezave na spletni aplikaciji.

V Odseku kode 3.3 je prikazana koda, ki povzroči vezavo rokovalnikov s povratnimi funkcijami (preko orodja `pageCallbacks`), nastavljanje lastnosti objekta `page` prek `get` ter `set` metod in zažene izvajanje dinamične spletne aplikacije.

```
pageCallbacks.startAllCallbacks(this);

this.page.get('settings', function (err, settings) {
    settings.loadImages = false;
    this.page.set('settings', settings, function () {
```

```
        this.page.open(this.currentUrl);
        setTimeout(this.evaluatePage.bind(this), this.waitTime);
    }.bind(this));
}.bind(this));
```

Odsek kode 3.3: Koda, ki kliče orodje za vezavo povratnih funkcij na rokovalnike, nastavi lastnosti izvajanja dinamične spletne aplikacije in požene izvajanje.

Pridobivanje podatkov o virih

Rokovalniki orodja PhantomJS nam omogočajo klice povratnih funkcij v primeru dogodkov. Dogodki, ki so za nas pomembni, so pošiljanje zahteve po virih (ang. `resource request`), prejemanje odgovora (ang. `resource response`), razne napake in še nekaj drugih. Preko teh dogodkov si lahko znotraj objekta `Page` shranjujemo vse, kar se dogaja z viri. Shranjujemo praktično vse, kar se da prek povratnih funkcij zajeti. Med te lastnosti spadajo identifikator vira, naslov zahteve, naslov odgovora, preusmeritve, uporabljena metoda, statusi ter glave protokola HTTP (`HyperText Transfer Protocol`), čas sprejema vira, napake, velikost vira, število prenesenih kosov (ang. `chunks`) ipd. Vse to se shrani kot objekt, ki predstavlja entiteto vira, in bo kasneje shranjen v PB MongoDB.

Pomembno je omeniti, da je pri virih zahteva z identifikacijsko številko 1 pravzaprav osnovna zahteva protokola HTTP, ki sproži nalaganje spletne aplikacije in prenos osnovnega dokumenta (ponavadi HTML). Zato so nekateri podatki prvega vira shranjeni tudi v entiteto obiska te spletne aplikacije.

V Odseku kode 3.4 je prikazan način vezave povratnih funkcij na rokovalnike v orodju `pageCallbacks`, možno pa je videti tudi vse uporabljene rokovalnike (vsi so našteti v dokumentaciji orodja [13]). V funkcijo je podan objekt, ki nosi ime `page` in je instanca razreda `Page`. Eden njegovih atributov pa je objekt `page`, ki ga generira orodje PhantomJS in predstavlja peskovnik v orodju.

```
module.exports.startAllCallbacks = function (page) {
    page.page.onError = onErrorCallback.bind(page);
    page.page.onUrlChanged = onUrlChangedCallback.bind(page);
    page.page.onLoadStarted = onLoadStartedCallback.bind(page);
    page.page.onLoadFinished = onLoadFinishedCallback.bind(page);
```

```
    page.page.onResourceError = onResourceErrorCallback.bind(page);
    page.page.onResourceRequested = onResourceRequestedCallback.bind(page);
    page.page.onResourceReceived = onResourceReceivedCallback.bind(page);
    page.page.onResourceTimeout = onResourceTimeoutCallback.bind(page);
};
```

Odsek kode 3.4: Vezava povratnih funkcij na rokovalnike orodja PhantomJS.

Pridobivanje povezav iz spletne aplikacije

Po preteku časa, podanega v parametru `waitTime`, od klica funkcije `page.open` se izvede t. i. evaluacija. To je funkcija `page.evaluate`, ki omogoča izvedbo JS (`JavaScript`) kode znotraj naložene spletne aplikacije. Tako lahko iz nje dobimo vse povezave. Povezave pridobivamo iz značk `a`, programskega jezika HTML, in njihovega atributa `href`. Ko imamo zbrane vse povezave, jih procesiramo z orodjem `urlProcess`. To orodje poskrbi za izločitev neustreznih povezav, kot so povezave, ki nas ne zanimajo, so neustrezne, nimajo navedenega protokola (`http://` ali `https://`) ali pa se nahajajo na seznamu neustreznih domen (`blacklist`). Poskrbi tudi za ustrezno procesiranje URL-jev, ki vsebujejo relativno pot znotraj trenutne spletne aplikacije, v absolutne naslove in za naslove s preveliko globino. Naknadno se poskrbi še za izločitev duplikatov spletnih naslovov že znotraj spletne aplikacije, saj so spletni naslovi shranjeni v PB kot entitete povezav, ki pa morajo biti unikatne. Na ta način lahko zmanjšamo število potrebnih poizvedb in hkrati izboljšamo zmogljivost spletnega pajka.

V Odseku kode 3.5 je prikazana evaluacija in zajemanje povezav v izvajani dinamični spletni aplikaciji.

```
this.page.evaluate(function () {
    var a = document.links;
    var links = [];
    for (var i = 0; i < a.length; i++)
        links.push(a[i].getAttribute('href'));

    return links;
}, function (errorEvaluate, results) {
    // obdelava naslovov URL
});
```

Odsek kode 3.5: Izvedba evaluacije v peskovniku, ki izvaja dinamično spletno

aplikacijo.

Shranjevanje podatkov

Po končanem pregledu spletne aplikacije sledi še shranjevanje podatkov v PB. To se izvede prek orodja `databaseSaveCollection`. Najprej se shranijo povezave. Če je zahtevano, se za vsako povezavo generira tudi hierarhija domen na enak način kot za obiske. V času testiranja smo ugotovili, da tako delovanje bistveno upočasni spletnega pajka zaradi zahtevne operacije generiranja domen že pri majhnem številu povezav. Če pa je število povezav dosti večje, pa je upočasnitev še toliko večja. Reference na vse povezave se dodajo v polje atributa `links` entitete obiska.

Ko se shranijo vse povezave, se začne še s shranjevanjem virov. Vsak vir je potrebno povezati z obiskom in gostiteljem virov. V primeru da gostitelj še ne obstaja, ga ustvarimo. Kot ime gostitelja virov se jemlje domena naslova vira. Gostitelj virov pri viru `http://www.google-analytics.com/ga.js` je tako `www.google-analytics.com`. Gostitelju virov se doda v polje `visits` tudi referenca na obisk.

Ko se shranjevanje uspešno izvede, se objekt `page` orodja PhantomJS zapre in izvajanje spletne aplikacije se sklene z uspešno preiskano spletno stranjo. Ob koncu se kliče ustrezno funkcijo upravljalca niti, ki določa nadaljne delovanje niti in celotnega spletnega pajka.

Poglavje 4

Upravljanje spletnega pajka

Pri razvoju spletnega pajka se je že pojavilo vprašanje o dobri aplikaciji za upravljanje spletnega pajka in obstoječi aplikaciji za ogled podatkov shranjenih v PB. Delno tudi zaradi pomanjkanja orodij, ki bi omogočala dobro vizualizacijo podatkov, smo se namerili izdelati lastno upravljalno orodje za našega spletnega pajka, ki bi vse to omogočalo. Orodje smo se odločili izdelati kot spletno aplikacijo.

Razvoj aplikacije za upravljanje spletnega pajka je potekal v štirih fazah. V prvi je bilo postavljeno ogrodje upravljalnega orodja, kot je okvir spletne aplikacije na strežniški strani, osnovna aplikacija na strani odjemalca in oblikovanje izgleda. V drugi fazi je sledilo postavljanje strukture za brskanje po podatkih glede na izbrani entitetni tip. Omogočen je bil tudi vpogled v vse podrobnosti posameznih entitet. V tretji fazi se je postavilo nekaj osnovnih analiz. O njihovi izdelavi govorimo v Poglavju 5. V četrti fazi pa je bil postavljen in izdelan vmesnik za zagon in nadzor spletnega pajka iz spletne aplikacije.

V tem poglavju je v začetku opisan razvoj upravljalnega orodja kot spletne aplikacije (4.1), v katerem govorimo o že omenjenem konceptu in prednosti programskega jezika JavaScript na strani odjemalca. Nadalje je opisan razvoj strežniškega sistema za podporo upravljalnemu orodju v obstoječem izvajalnem okolju spletnega pajka (4.2). Na koncu je predstavljeno samo delovanje upravljalnega orodja (4.3).

4.1 Razvoj spletne aplikacije na strani odjemalca

Stran odjemalca (ang. `client-side`) v spletnih aplikacijah večinoma predstavljajo spletni brskalniki oziroma aplikacije, ki so jih sposobne zagnati (v primeru spletnega pajka je to peskovnik, ki deluje znotraj orodja PhatnomJS). Ti ob izvajanju spletnih aplikacij sprožajo zahteve po virih, ki omogočajo njihov prikaz in izvajanje (Poglavje 2.2). Kot je že omenjeno v Poglavju 2.2.2, so se v zadnjem obdobju spletni brskalniki izredno razvili, z njimi pa so se razvili tudi pogoni za izvajanje kode JavaScript, kar je močno vplivalo na uveljavitev tega programskega jezika. Omenili smo že, kako se je to pokazalo v razvoju strežniških okolij, močno pa je vplivalo tudi na razvoj na strani odjemalca spletnih aplikacij. V zadnjih časih se večji del generiranja in izvajanja vsebin za prikaz spletnih aplikacij seli na brskalnike, pri čemer močno izkoriščajo vse prednosti, ki jih nudita AJAX in vedno hitrejši pogoni za JavaScript.

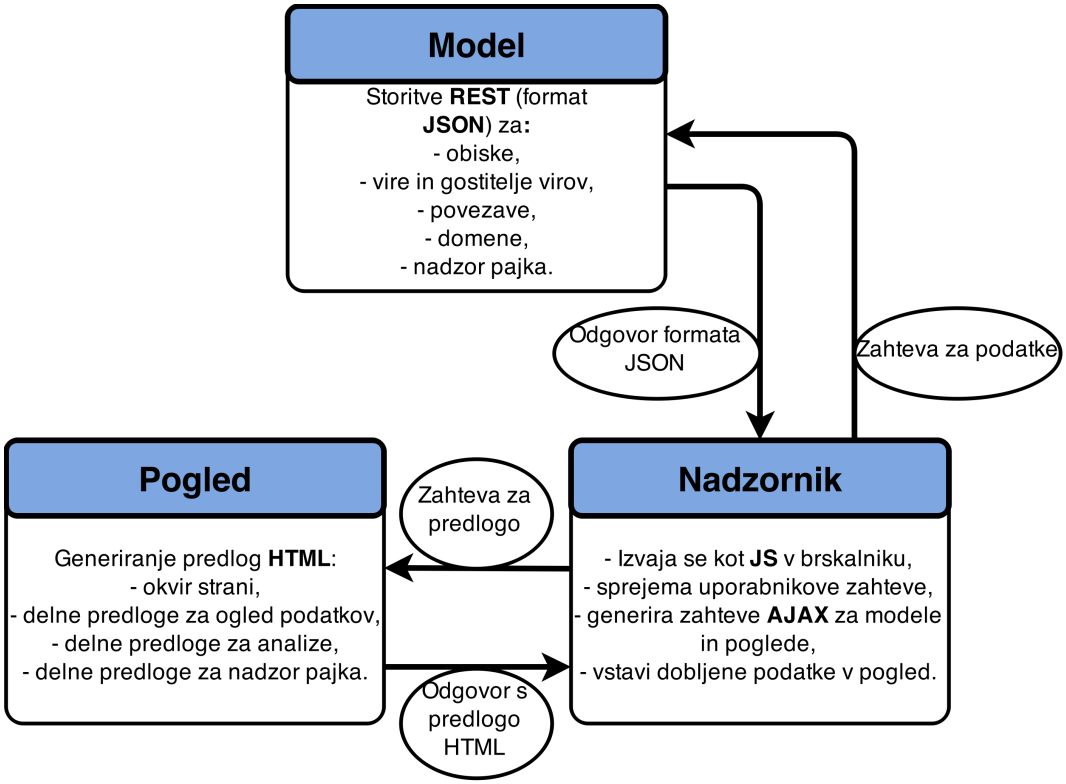
V tem podpoglavju je predstavljen koncept, ki se uveljavlja na področju razvoja spletnih aplikacij na strani odjemalca (4.1.1), in orodje AngularJS (4.1.2), s katerim smo realizirali odjemalsko stran našega upravljalnega orodja.

4.1.1 MVC na strani odjemalca

MVC (`Model, View, Controller`) je načrtovalski vzorec (ang. `design pattern`), ki spletno aplikacijo deli na tri nivoje:

- Model (ang. `model`) – predstavlja podatke.
- Pogled (ang. `view`) – predstavlja uporabniški vmesnik oziroma izgled aplikacije.
- Nadzornik (ang. `controller`) – predstavlja povezavo med pogledom in modelom ter zagotavlja interakcijo z uporabnikom.

Shema MVC na strani odjemalca za primer upravljalnega orodja za spletnega pajka je vidna na Sliki 4.1, kjer je specifično za naš primer prikazano, kako MVC deluje. Na strežniški strani je ta načrtovalski vzorec uveljavljen že kar nekaj časa,



Slika 4.1: Shema načrtovalskega vzorca MVC na strani odjemalca za primer upravljalnega orodja za spletnega pajka.

predvsem v tehnologijah Ruby on Rails in Django. Načrtovalski vzorec MVC na strani odjemalca za JavaScript pa je veliko mlajši. Strežnik tako ne generira več celotne vsebine v en dokument HTML, ki ga pošlje brskalniku, ampak se vsebina zgradi šele v brskalniku s pomočjo JavaScipta (nadzornik), predlog HTML (pogled) in JSON ali datotek XML (model). Strežnik servira zgolj statične vsebine – predloge HTML, slike, skripte (datoteke JS), oblikovne datoteke CSS (`Cascading Style Sheets`) in podobno, ki jih ne prilagaja vsakemu uporabniku posebej. Streže tudi podatke, ki jih črpa iz PB, odjemalcu pa jih pošilja v formatu JSON ali XML, ponavadi prek storitev REST. Na ta način se poskušajo razbremeniti strežniki, ki sicer z generiranjem vsebine porabljajo precej resursov. Strežnikom tudi ni potrebno več hraniti stanja o uporabniku, saj za vse to poskrbi nadzornik na strani odjemalca.

4.1.2 AngularJS

AngularJS je strukturirano ogrodje za razvoj dinamičnih spletnih aplikacij. Izvaja se na strani odjemalca, v spletnem brskalniku, in izkorišča vse prednosti načrtovalskega vzorca MVC na strani odjemalca. Nudi močna orodja za povezovanje podatkov (ang. `data binding`), vstavljanje odvisnosti (ang. `dependency injection`) in delo s strukturo DOM. V veliki meri uporablja AJAX in omogoča izrabo vseh operacij CRUD, največkrat prek storitev REST [9].

V diplomski nalogi je bil AngularJS uporabljen za izgradnjo spletne aplikacije, ki nudi vpogled v podatke, pridobljene s spletnim pajkom. Z njim je bilo realizirano brskanje po podatkih, ogled vseh podrobnosti, ogled ter generiranje analiz in zagon spletnega pajka direktno iz spletne aplikacije. Uporabljena je bila verzija 1.2.19.

4.2 Razvoj spletne aplikacije na strežniški strani

Ob uporabi in pregledu ogrodja Node.js smo hitro naleteli na modul `express`. Modul ponuja osnovno strukturo in funkcionalnosti, ki jih spletni strežnik potrebuje za svoje delovanje [2]. Zaradi priporočil, uporabnosti, enostavnosti in dobrih izkušenj z delom na ogrodju Node.js smo se odločili, da bomo modul uporabili tudi pri naši

aplikaciji. Strežniški del mora biti razvit tako, da podpira arhitekturo MVC na strani odjemalca. Z ustrezno postavitvijo modula `express` (4.2.1) smo strežnik zasnovali tako, da nudi statične vsebine kot tudi dinamično generirane podatke. Pri streženju podatkov smo uporabili storitve REST (4.2.2).

4.2.1 Uporaba modula express v izvajalnem okolju

Pri vgrajevanju modula `express` v izvajalno okolje Node.js smo želeli združiti spletni strežnik za podporo delovanju upravljalnega orodja z ostalo izvorno kodo spletnega pajka. Ugotovili smo, da je bila datotečna struktura kode spletnega pajka ustrezna in je dopuščala združevanje s kodo za delovanje spletnega strežnika za spletno aplikacijo. Po pregledu nekaj vodičev za skupno delovanje modula `express` v ogrodju Node.js in orodja AngularJS smo v obstoječo strukturo izvorne kode dodali naslednje mape:

- `public` – tu so hranjene vsebine, kot so skripte JS, datoteke CSS, slike in pisave, torej datoteke, ki na strežniški strani ne potrebujejo dodatnega procesiranja.
- `routes` – v direktoriju se nahajajo skripte, ki obdelujejo zahteve po različnih resursih, ki potrebujejo procesiranje s strani strežnika. V diplomski nalogi smo tu notri vključili storitve, ki zgradijo predloge HTML in ustrežejo zahtevam po podatkih prek storitev REST.
- `views` – v tem direktoriju se nahajajo neobdelane predloge HTML.

Za gradnjo predlog HTML smo se odločili preizkusiti jezik Jade, ki je nekakšna alternativa HTML-ju, zelo pa se priporoča ob modulu `express`. Uporablja enake značke, nekoliko drugačna je le sintaksa, ki omogoča večjo preglednost in hitrejše delo. Ob zahtevi za določeno vsebino HTML se zahteva poveže z ustrezno predlogo Jade, ki se na strežniku pretvori v jezik HTML. Kot odgovor se pošlje datoteka s pretvorjeno vsebino. Primer pretvorbe je viden na Sliki 4.2. Naj še posebej izpostavimo, da se te vsebine zgenerirajo za vsakega uporabnika enako. Procesiranje je potrebno le zaradi dela v jeziku, ki nudi drugačno sintakso kot HTML in ne generira vsakemu uporabniku posebej prilagojene vsebine.

```
!!! 5
html
    head
        meta(charset='utf8')
        title Web Crawler
        link(rel='stylesheet', href='/css/bootstrap.min.css')
        script(src='js/lib/jquery.min.js')
    body
        h2 Browse Domains
        p Domains represent hierarchy of all domains which...
        div(class='panel panel-default')
            div(class='panel-heading')
                h4 Domain Hierarchy
            div(class='panel-body', id='tree')
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf8">
    <title>Web Crawler</title>
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <script src="js/lib/jquery.min.js"></script>
</head>
<body>
    <h2>Browse Domains</h2>
    <p>
        Domains represent hierarchy of all domains which...
        <div class="panel panel-default">
            <div class="panel-heading">
                <h4>Domain Hierarchy</h4>
            </div>
            <div id="tree" class="panel-body"></div>
        </div>
</body>
</html>
```

Slika 4.2: Primer pretvorbe iz jezika Jade v HTML.

Z Jadeom smo torej zgradili okvir (ang. `layout`) strani in delne predloge, ki se preko orodja AngularJS vključujejo v sam okvir. To omogoča asinhrono nalaganje delnih predlog, medtem ko okvir ostaja enak. Okvir na odjemalcu zahteva nalaganje vseh potrebnih knjižnic za delovanje orodja AngularJS (`angular`, `angular-routes` in `angular-resources`), knjižnic za JS (`jQuery`, `Bootstrap`), datotek CSS in skript, ki določajo delovanje orodja. Za izgled spletne aplikacije smo uporabili znano knjižnico za CSS in JS `Bootstrap`, ki je sicer v večji meri namenjena podpori mobilnim spletnim aplikacijam [1]. Knjižnica nam je ustrezala zaradi vseh komponent, ki jih nudi in z njimi omogoča hitro izgradnjo izgleda spletne aplikacije. Z enostavnim spreminjanjem definiranih razredov pa lahko pridemo do prilagojenega izgleda spletne aplikacije za naše potrebe.

Vsa konfiguracija spletnega strežnika se hrani v eni skripti. V njej modulu `express` podamo vse relativne poti, do katerih dovolimo dostop, in povratne metode, ki se izvedejo ob prejeti zahtevi na ta spletni naslov. Za omenjene vsebine HTML na primer podamo funkcije, ki zgenerirajo vsebino HTML iz predlog Jade, za zahteve po podatkih prek storitev REST pa funkcije, ki komunicirajo s PB in zgenerirajo odgovor formata JSON.

Realizacija nudenja delnih predlog HTML (ki se nahajajo v podmapi `partials` direktorija `views`) je vidna v Odseku kode 4.1.

```
app.get('/partials/:name', function (req, res) {
```

```
    var name = req.params.name;
    res.render('partials/' + name);
});
```

Odsek kode 4.1: Relativna pot, ki proži generiranje vsebine HTML iz predlog Jade.

4.2.2 Realizacija storitev REST

Storitve REST smo se odločili uporabiti za posredovanje potrebnih podatkov iz PB MongoDB orodju AngularJS. V Poglavju 4.1.1 je razložen načrtovalski vzorec MVC na strani odjemalca. V diplomski nalogi so podatkovni del načrtovalskega vzorca (model) nudile storitve REST. Za vseh pet entitetnih tipov smo v začetku kreirali nekaj osnovnih storitev, ki nudijo najbolj osnovne podatke. Sčasoma smo storitve dograjevali v odvisnosti od potreb, ki so se pokazale ob gradnji spletne aplikacije. S storitvami REST kreiramo odgovore v formatu JSON. Ta se nam je zdel daleč najbolj primeren, saj gre pravzaprav za notacijo objekta v jeziku JS (ang. `JavaScript Object Notation`). Tudi odgovore pri poizvedbah v PB MongoDB je sila enostavno pretvoriti v niz znakov, ki predstavljajo notacijo JSON.

Storitve REST so arhitektura, ki v večini primerov uporabljajo protokol HTTP za upravljanje s podatki prek naslovov URI (`Uniform Resource Identifier`) in jih lahko nudimo v več formatih (XML, JSON, tekst, HTML ipd.). Ponavadi uporabljamo različne metode protokola HTTP za različno manipulacijo s podatki. Metode lahko glede na delo s podatki in operacijami CRUD v PB prevedemo na naslednji način:

- `POST` – ustvarjanje novih vnosov v PB (operacija `Create`).
- `GET` – branje vnosov iz PB (operacija `Read`).
- `PUT` – posodobitev vnosov v PB (operacija `Update`).
- `DELETE` – brisanje vnosov iz PB (operacija `Delete`).

V diplomski nalogi prek storitev REST omogočamo zgolj branje podatkov preko storitev REST in podajanje le-teh le v formatu JSON. Prek metode `POST` protokola

HTTP se zgolj sprejmejo parametri pri zagonu spletnega pajka prek spletne aplikacije, kar je opisano v Poglavju 4.3.2. Storitve REST je v modulu `express` zelo enostavno realizirati. Na enak način podamo pot, po kateri se zahtevajo podatki, v povratni funkciji pa izvedemo poizvedbo v bazo. Težji del je bil tu torej edino pri nekaterih poizvedbah, ki so zahtevale malo več kreativnosti in uporabe nekoliko bolj zahtevnih poizvedb, kot je agregacija. Primer take poizvedbe in generiranja odgovora je prikazan v Odseku kode 4.2.

```
app.get('/rest/resources/unique-url', function (req, res) {
    var aggregate = Resource.aggregate([
        { $group: { _id: '$request_url', count: { $sum: 1 } } },
        { $sort: { count: -1 } },
        { $limit: 20 }
    ]);
    aggregate.allowDiskUse(true).exec(function (err, results) {
        if (!err && results.length > 0) {
            res.writeHead(200, { ContentType: 'application/json' });
            res.write(JSON.stringify(results));
            res.end();
        } else {
            res.writeHead(404, { ContentType: 'application/json' });
            res.end();
        }
    });
});
```

Odsek kode 4.2: Izvajanje agregacije na podatkih ob prejemu zahteve `GET` pri storitvi REST.

Poizvedba najprej grupira entitete virov v PB po spletnem naslovu zahteve (`request_url`) in prešteje število zahtev z enakim naslovom. V naslednjem koraku rezultate uredi padajoče po številu zahtev na naslov. V zadnjem koraku omeji število rezultatov na 20. S funkcijo `allowDiskUse` se omogoči uporaba diska med poizvedbo zaradi velikega števila entitet v zbirki.

4.3 Delovanje upravljalnega orodja

V tem poglavju so opisane funkcionalnosti, ki jih nudi orodje za upravljanje spletnega pajka. V prvem poglavju (4.3.1) je predstavljena funkcionalnost brskanja po

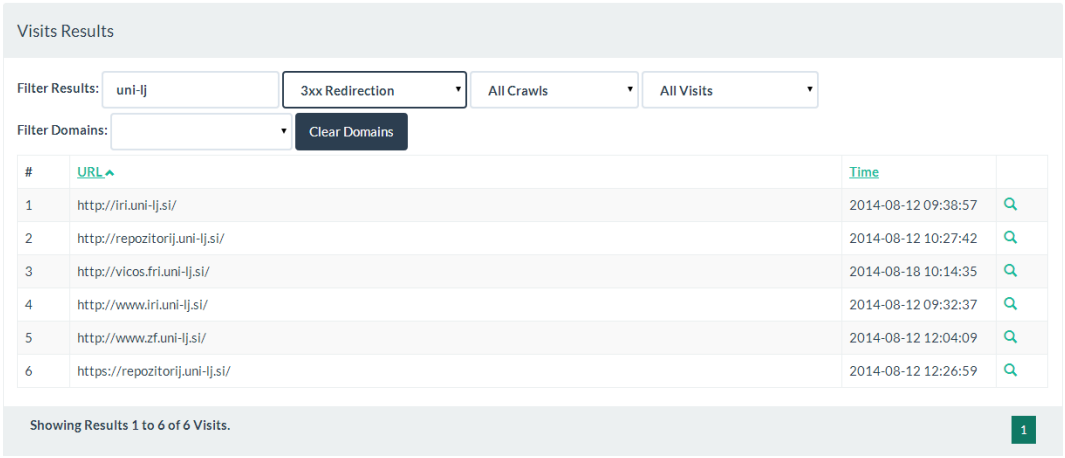
podatkih, prek katere si lahko ogledamo vse podrobnosti o shranjenih entitetah in njihovih relacijah. Tako je omogočen podroben ogled in analiza povezovanja dinamičnih spletnih aplikacij med seboj in z zunanjimi ponudniki storitev. V drugem poglavju pa je predstavljen zagon in nadzor našega spletnega pajka (4.3.2). Orodje poleg omenjenih funkcionalnosti omogoča tudi prikaz analiz, ki so izdelane nad vsemi podatki v podatkovni bazi. Te so predstavljene skupaj z rezultati vzorčnega sprehoda v Poglavju 5.

4.3.1 Brskanje po podatkih

V spletni aplikaciji je omogočeno brskanje po entitetah obiskov, virov, gostiteljev virov in domenah. Za brskanje po enitetah povezav se nismo odločili zaradi tega, ker je ta primarno namenjena vrsti FIFO za pridobivanje spletnih naslovov za preiskovanje spletnega pajka. Prikaz brskanja po domenah bomo predstavili ločeno od predstavitve brskanja po ostalih domenah, saj smo z njimi gradili hierarhijo in so zato prikazane temu ustrezno – z drevesno strukturo in ne s seznami.

Brskanje po seznamih obiskov, virov in gostiteljev pri vsakemu omogoča filtriranje prikazanih rezultatov glede na različne parametre. Še največ možnosti je pri obiskih, kjer se lahko filtrira naslove URL, statuse protokola HTTP, obiske z napakami pri sprehajanju in obiske, ki vsebujejo napake na spletni aplikaciji (npr. v konzoli spletnega brskalnika), omogočeno pa je tudi filtriranje po domenah. Možno je tudi razvrščanje po različnih atributih. Izgled seznama je viden na Sliki 4.3.

Pri vseh seznamih je možen tudi ogled podrobnosti entitet. Pri ogledu se prikažejo vsi osnovni podatki, ki so shranjeni kot atributi, možen pa je tudi ogled relacij, če te seveda obstajajo. Pri obiskih je možen ogled vseh povezav, najdenih na strani, in ogled vseh virov, ki so se na aplikacijo naložili. Pri gostiteljih virov pa si je možno ogledati, katere spletne aplikacije oziroma obiski so nalagali vire s tega gostitelja, možno pa si je tudi ogledati seznam domen, ki so se povezovale s tem virom. Pri podrobnostih virov je možen ogled obiskov, kamor so se viri z enakim naslovom prenašali. Na vseh straneh o podrobnostih entitet so prikazani tudi grafi, ki so za dano entiteto lahko zanimivi. Grafi so zgrajeni z orodjem Flot, ki je knjižnjica za JS in omogoča izris grafov glede na podane podatke ter parametre za izgled. Primer



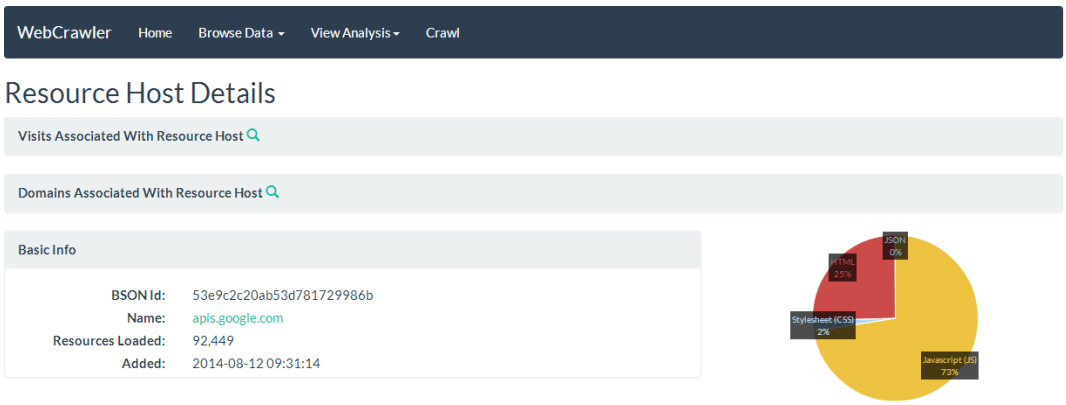
Slika 4.3: Grafičen izgled vmesnika spletne aplikacije za prikaz seznama obiskov, ki je filtriran na naslove URL, ki vsebujejo niz `uni-lj` in kodo statusa protokola HTTP med 300 in 399 (preusmeritev).

prikaza podrobnosti o gostitelju virov je na Sliki 4.4.

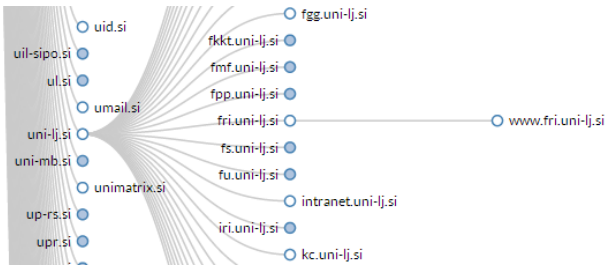
Omenjeno je bilo, da so domene prikazane drugače. Ker je zbirka domen namenjena grajenju hierarhije domen, je tudi njen prikaz temu primeren. Pri ogledu hierarhije domen si je tako možno ogledati kar drevesno strukturo, ki prikazuje povezovanje različnih nivojev domen (Slika 4.5). S klikom na posamezno vozlišče se nam prikažejo nasledniki tega vozlišča, če jih ima. Drevo je zgrajeno z orodjem d3, ki je knjižnjica za JS.

4.3.2 Zagon spletnega pajka iz upravljalnega orodja

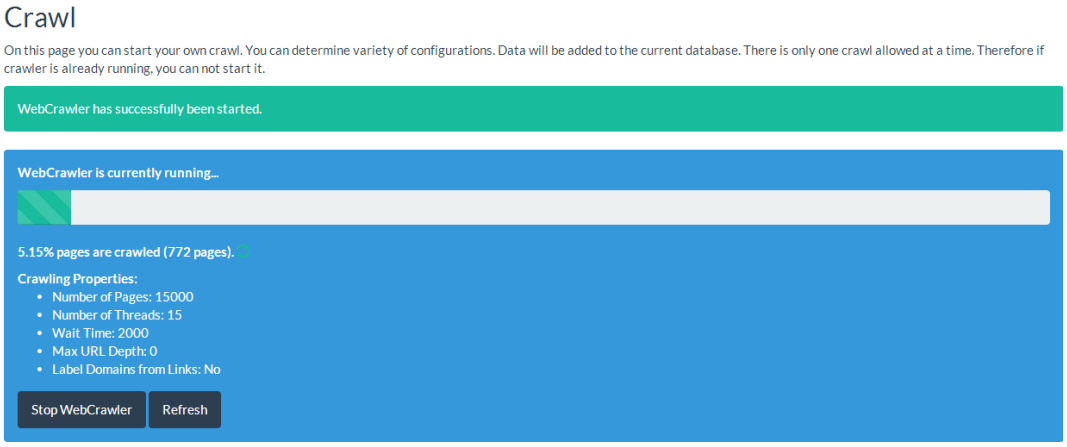
Zadnja stvar, ki smo jo realizirali v spletni aplikaciji, je bila zagon spletnega pajka iz upravljalnega orodja. Najprej je bilo potrebno na strežniški strani spletne aplikacije urediti podporo v ta namen. V modulu `express` sta se odprli dve dodatni poti – ena za zagon spletnega pajka, druga pa za ustavitev. Poti sprejmeta metodo `POST` protokola HTTP, saj je potrebno za zagon prejeti parametre, ki določajo delovanje spletnega pajka. Ti parametri se v spletni aplikaciji podajo v obliki običajnega spletnega obrazca (ang. `form`). Povratna funkcija, ki se izvede ob zahtevi za zagon



Slika 4.4: Grafičen izgled vmesnika spletne aplikacije za prikaz podrobnosti o gostitelju virov.



Slika 4.5: Grafični izgled vmesnika spletne aplikacije za prikaz hierarhije domen.



Slika 4.6: Izgled prikaza napredka spletnega pajka, zagnanega z upravljalnim orodjem.

pajka, najprej preveri ustreznost parametrov. Če ti ustrezajo, jih shrani v t. i. konfiguracijski objekt, ki se v formatu JSON shrani v datoteko na strežniku.

Spletni pajek se zažene z uporabo funkcije `spawn` ogrodja Node.js. Funkcija omogoča zagon procesa, ki postane otrok trenutnega izvajalnega okolja. Za zagon spletnega pajka se generira nova instanca izvajalnega okolja Node.js, zažene pa se skripta, ki prebere datoteko, v katero je zapisan konfiguracijski objekt v formatu JSON. Skripta zapis v datoteki razčleni (ang. `parse`), s pridobljenimi parametri pa kreira nov objekt `Phantom`, ki predstavlja tudi zagon spletnega pajka. Starševski proces lovi ter izpisuje v konzolo vse izpise spletnega pajka in si shranjuje njegov napredek v globalno spremenljivko. Njena vrednost je dostopna tudi prek storitve REST, da lahko spletna aplikacija spremlja napredek sprehajanja.

Ker se spletni pajek zaganja iz spletne aplikacije, je uvedenih nekaj omejitev. Maksimalno število dovoljenih niti je 20, maksimalno število spletnih naslovov za preiskovanje pa je omejeno na 1000 aplikacij na nit. Prav tako lahko na strežniku teče zgolj en otroški proces, kar pomeni eno instanco spletnega pajka.

Na strani odjemalca spletne aplikacije je bilo dela za realizacijo zagona spletnega pajka bistveno manj. Potrebno je bilo dodati obrazec in urediti prikaz, ki ponazarja napredek sprehoda (Slika 4.6).

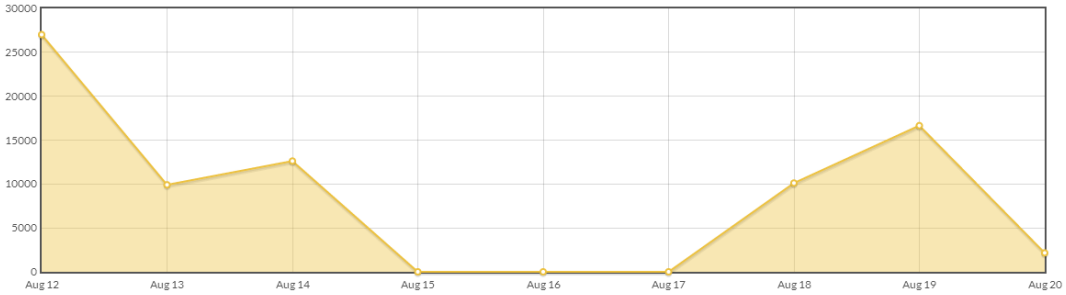
Poglavje 5

Izvajanje analiz in rezultati vzorčnega sprehoda

V prvem poglavju Poglavja 5 je predstavljen način izvajanja analiz, ki se izvajajo s pomočjo upravljalnega orodja za spletnega pajka (5.1). V zadnjem Poglavju 5.2 pa so predstavljeni rezultati vzorčnega sprehoda s 100.000 dinamičnimi spletnimi aplikacijami, s katerim smo pridobili vzorčno podatkovno bazo, ki nudi osnovo za izvajanje ter nadgrajevanje analiz in nadaljnje raziskovanje na področju problematike dinamičnih spletnih aplikacij.

5.1 Izvajanje analiz

Z izvajanjem analiz smo želeli pridobiti posplošen vpogled nad podatki v podatkovni bazi. Analize nam pokažejo rezultate sprehajanja, prek katerih imamo možnost generalnega vpogleda v pridobljene podatke, in so nekakšen kazalnik uspešnosti ter ustreznosti podatkov, ki smo jih zajeli. Analize se ves čas dopolnjujejo, saj šele skozi čas dobivamo nove povratne informacije o dinamičnih spletnih aplikacijah in novih skupinah podatkov, ki bi nas lahko zanimali. V tem poglavju je zato predstavljena splošna ideja izvedbe analiz, kar pomeni predstavitev narejenih analiz in njihovega načina izvedbe. Izdelane analize predstavljajo infrastrukturo za razvoj novih analiz, ki bi jih v nadaljnjih raziskavah o spletnih aplikacijah še potrebovali. Za pregled

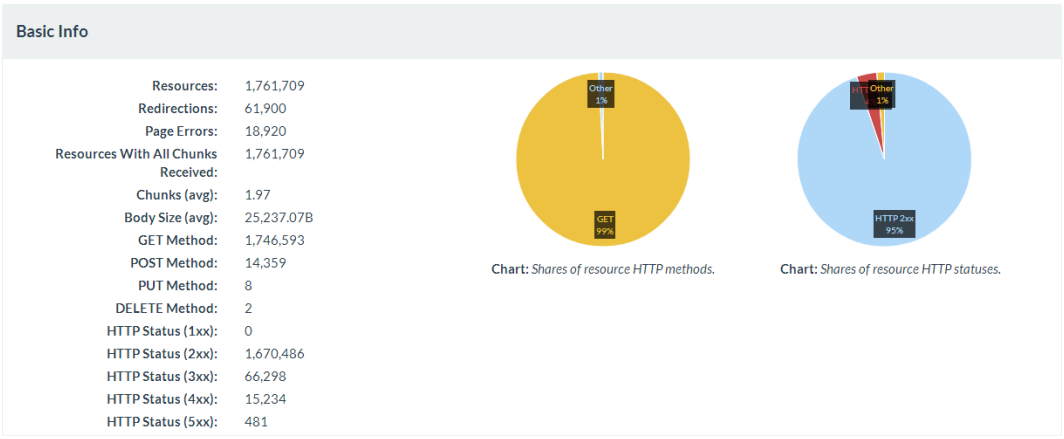


Slika 5.1: Graf analize števila preiskanih spletnih aplikacij po dnevih.

in analizo podrobnosti o entitetah ter njenih relacijah pa ustreza možnost brskanja po podatkih prek upravljalnega orodja za spletnega pajka in je opisana v Poglavju 4.3.1.

Izdelava analiz je izredno zahtevna, čeprav se morda na prvi pogled ne zdi tako. Problem se pojavlja predvsem v zahtevnosti nekaterih poizvedb pri velikem številu dokumentov v PB. Potrebno je vzeti v zakup, da nekatere poizvedbe enostavno potrebujejo več časa za izvedbo, v največji meri pa je to odvisno od števila entitet, ki so v poizvedbo vključene, in števila ter zahtevnosti operacij, ki se nad njimi izvajajo. Izdelavo analiz smo razdelili na 3 sklope – splošno analizo, analizo obiskov in analizo virov ter gostiteljev. Splošna analiza prikaže število dokumentov v vsaki zbirki v PB in graf, ki prikazuje delovanje spletnega pajka skozi čas. To pomeni, da prikaže analizo števila preiskanih spletnih aplikacij po dnevih in jo izriše v graf (Slika 5.1).

Analiza virov in gostiteljev prikazuje najprej splošne informacije (Slika 5.2), kot so število virov, preusmeritev, napak na virih, povprečno število prenesenih kosov ter velikosti virov in števila pojavitev posameznih metod ter statusov protokola HTTP. V nadaljevanju prikaže analizo 20 gostiteljev virov, s katerih se je največ prenašalo, 20 največkrat prenesenih virov (glede na naslov zahteve), analizo prejetih virov glede na različne tipe MIME (`Multipurpose Internet Mail Extension`) vrednosti in 20 največjih virov po velikosti. Vse te analize so dober pokazatelj ustreznosti sprehajanja glede na ciljne spletne aplikacije, ki nas zanimajo. Prek njih smo tudi med izvedbo vzorčnega sprehoda spremljali ustreznost sprehajanja in prek rezultatov



Slika 5.2: Prikaz osnovnih podatkov pri analizi virov in gostiteljev virov na spletni aplikaciji.

prenesenih virov in najpogosteje uporabljenih gostiteljev virov uravnavali parametre zagona spletnega pajka.

Pri analizi obiskov je prikazanih 20 obiskov z največ povezavami in 20 obiskov z največ naloženimi viri. Poleg tega so zopet prikazane splošne informacije, kot so povprečno število povezav (tako izven kot znotraj spletne aplikacije), povprečno število zahtevanih virov, število napak pri sprehajanju ter napak na spletni aplikaciji in število posameznih statusov protokola HTTP spletnih aplikacij. Analiza obiskov služi vpogledu v najbolj ekstremne dinamične spletne aplikacije z največkrat prenesenimi viri in največ vsebovanimi povezavami. Skozi razvoj spletnega pajka in faze testiranja različnih parametrov zagona so se pokazale skoraj zastrašujoče številke v več tisoč prenesenih virih in najdenih povezavah na posamezno dinamično spletno aplikacijo.

5.2 Rezultati vzorčnega sprehoda

Ko smo dosegli zadostno stabilnost v fazi razvoja spletnega pajka, smo izvedli večji sprehod na okoli 100.000 spletnih aplikacijah. Na ta način smo poskušali dobiti jasnejšo sliko o tem, kaj se dogaja z viri in s povezovanjem spletnih aplikacij, predvsem

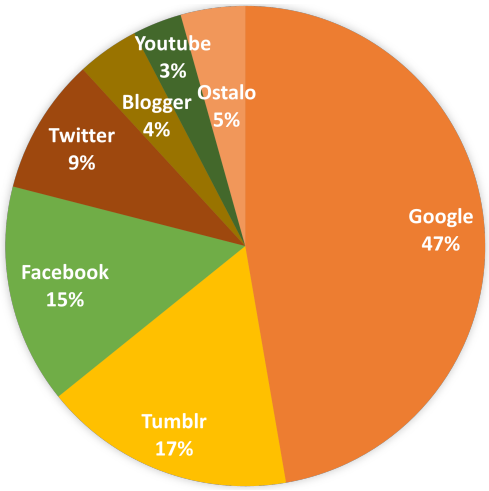
pa dobiti neko vzorčno PB, ki bo omogočala izdelavo analiz in podlago za nadaljnje raziskave. Zanimalo nas je tudi, kateri so najbolj uporabljani ponudniki virov in kateri viri se največ prenašajo. Spletnega pajka smo poganjali nekaj dni in vsak dan preiskali po več deset tisoč spletnih aplikacij. Kar nekaj časa smo potrebovali, da smo našli ustrezno konfiguracijo za pridobivanje relevantnih rezultatov. Problem se je pojavil predvsem v spletnih aplikacijah, ki imajo veliko notranjih povezav. Zaradi takih aplikacij in vrste FIFO za prevzemanje spletnih naslovov za preiskovanje je spletni pajek porabil občutno preveliko časa za preiskovanje podstrani določenih spletnih aplikacij in so zato rezultati analiz dajali netočne rezultate. Za boljše rezultate smo uvedli omejitev globine naslova URL na 0. To praktično pomeni, da je vsak obisk spletne aplikacije predstavljal zgolj obisk njenega osnovnega spletnega naslova, kot je na primer `http://www.domena.si/`.

V Tabeli 5.1 je prikazanih pet gostiteljev, s katerih se je prenašalo največ virov. Hitro lahko ugotovimo, da so očitno storitve podjetja Google najbolje zaželjene, saj sta med petimi gostitelji kar dva njihova, od tega je eden prvouvrščen. Pri največkrat zahtevanih virih pa so med prvimi petimi prav vsi viri naloženi iz enega izmed gostiteljev podjetja Google. Med njimi je na petem mestu tudi skripta, ki omogoča delovanje znane storitve Google Analytics za beleženje obiska spletnih aplikacij. Če pogledamo graf prvih 20 gostiteljev virov in njihove deleže prenosov (Slika 5.3), ugotovimo, da kar 47% prenosov predstavlja gostitelje podjetja Google. Če k temu prištejemo še deleže Youtubea, pa delež naraste na točno polovico. Sledijo mu Tumblr, Facebook in Twitter. Tipi datotek, ki so se prenašali, so se prešteli glede na specificiran tip MIME (`Multipurpose Internet Mail Extension`). Graf deležev je viden na grafu na Sliki 5.4. Kar polovica prenesenih je bilo izvajalnih datotek programskega jezika JS, četrtina je bila datotek HTML, petina pa oblikovnih datotek CSS. Med ostalimi so zajeti še tipi datotek, ki so nosile pisave (1,9%), datoteke z zapisom JSON (1,3%), čisti tekst (0,8%) in datoteke tipa XML (0,1%).

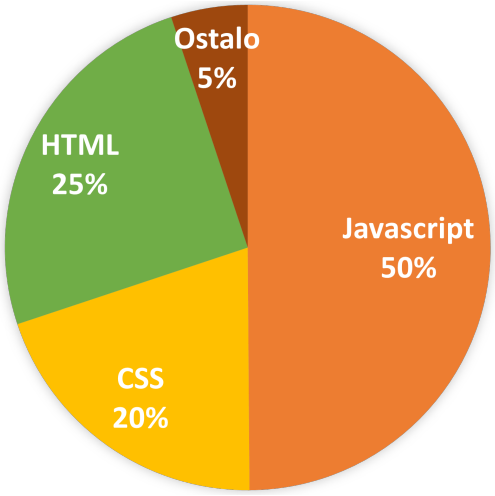
Zanimivo si je pogledati tudi spletne aplikacije z največ prenesenimi viri v Tabeli 5.2. Tu je potrebno omeniti, da gre pri prvi spletni aplikaciji za preusmerjeno spletno aplikacijo, ki se preusmerja sama vase, kar privede do cikla. Zato tudi toliko virov. Vse ostale pa so popolnoma verodostojne. Ugotovili smo, da gre večinoma za aplikacije, ki dokaj pogosto posodabljajo svojo vsebino, zato se je v času

#	Gostitelj vira	Št. prenesenih virov
1	apis.google.com	149.478
2	assets.tumblr.com	57.998
3	www.facebook.com	50.307
4	platform.twitter.com	39.474
5	www.google.com	36.402

Tabela 5.1: 5 gostiteljev virov, s katerih se je preneslo največ virov.



Slika 5.3: Graf deležev podjetij, katerih gostitelji virov so zabeležili največ prenesenih virov. Deleži so narejeni za 20 najvišje uvrščenih gostiteljev.



Slika 5.4: Graf deležev tipov datotek glede na določen tip MIME v glavah protokola HTTP.

nalaganju aplikacije zgodilo tako veliko zahtev. Veliko zahtev je tudi preusmerjenih in verjetno tudi to doprinese k nekaj več generiranim zahtevkom po virih. Vseeno so nas rezultati presenetili, saj je prenesenih virov pri spletnih aplikacijah veliko več od pričakovanega. Čeprav smo imeli omejitev globine naslova URL nastavljeno na 0, se je našlo nekaj spletnih aplikacij s presenetljivo velikim številom povezav. Prva jih je imela kar 2.438, še dve naslednji pa sta jih imeli čez 1.000.

Naj na tem mestu povemo še nekaj o porabi resursov spletnega pajka in na

#	URL spletne aplikacije	Št. prenesenih virov
1	http://brocprozac.com/	1.763
2	http://f24.my/	1.386
3	http://kirhion.tumblr.com/	1.301
4	http://www.orioles101.com/	575
5	http://www.techeblog.com/	519

Tabela 5.2: 5 spletnih aplikacij z največjim številom prenesenih virov.

splošno o zmogljivosti naše implementacije spletnega pajka. Izvajalno okolje Node.js v času izvajanja spletnega pajka porabi nekje od 50 do 250 megabajtov pomnilnika RAM, pri CPU-ju (`Central Processing Unit`) pa nismo zasledili večjih porab razen manjših skokov. Vsaka instanca orodja PhantomJS porabi od 30 do 120 megabajtov pomnilnika in je procesorsko nekoliko bolj obremenjujoča. Še najbolj pa sistem obremenjuje proces `mongod.exe`, ki predstavlja PB MongoDB. Večja poraba se zazna predvsem takrat, ko se shranjujejo podatki v PB. Pri večih nitih spletnega pajka se to dogaja zelo pogosto, zato je PB najbolj obremenjena komponenta spletnega pajka.

Zmogljivost delovanja našega spletnega pajka se je izkazala za boljšo od pričakovanj. Seveda je bila daleč od dosega zmogljivosti opisanih spletnih pajkov v Poglavju 2.1, vendar se je potrebno zavedati, da so to izredno napredni sistemi, ki delujejo na zmogljivih porazdeljenih sistemih, preiskovati pa so sposobni več milijonov spletnih aplikacij dnevno. Naš spletni pajek je na računalniku s sistemom Microsoft Windows 7, ki vsebuje disk SSD, 8 gigabajtov pomnilnika RAM in procesor Intel i7 z osmimi jedri, za 10.000 preiskanih spletnih aplikacij porabil dobri dve uri in pol, kar povprečno pomeni 930 milisekund na spletno aplikacijo. V zadnji fazi razvoja pa je spletni pajek deloval v virtualnem računalniku, ki je nameščen na strežniku Laboratorija za integracijo informacijskih sistemov na Fakulteti za računalništvo in informatiko. Nameščen je bil operacijski sistem Microsoft Windows Server 2012, računalniku pa je bilo dodeljenih 32 gigabajtov pomnilnika RAM, procesor Intel Xeon s štirimi jedri in običajni trdi disk. Za 10.000 preiskanih spletnih aplikacij je potreboval 1 uro in 16 minut, kar pomeni povprečno 460 milisekund na aplikacijo. Toliko boljšo zmogljivost na strežniku laboratorija je možno pripisati večjemu pomnilniku RAM, kasnejši fazi razvoja, ko je bil spletni pajek že veliko bolj optimiziran, in veliko boljši ter hitrejši fakultetni internetni povezavi v primerjavi s počasno domačo mrežo.

Poglavje 6

Sklep

V diplomski nalogi je bil uspešno zasnovan in razvit spletni pajek za analizo dinamičnih spletnih aplikacij. Razviti spletni pajek se razlikuje od klasičnih spletnih pajkov, ki jih zanima semantika spletnih aplikacij večinoma za potrebe gradnje iskalnih indeksov spletnih iskalnikov. Klasični pajki zato za naše analize niso ustrezni. Našega spletnega pajka smo zasnovali tako, da dinamične spletne aplikacije izvaja v peskovniku znotraj navideznega spletnega brskalnika. Peskovnik nam je omogočil vpogled v celotno delovanje spletne aplikacije s sledenjem dogodkom, ki se odvijajo med izvajanjem spletne aplikacije. S tem je možno spremljanje nalaganja virov, ki jih aplikacija potrebuje za svoje delovanje. Pri izvajanju so najbolj kritični viri, ki vsebujejo izvajalne datoteke JavaScript, saj se prek njih povezujejo spletne aplikacije, obenem pa omogoča zajemanje uporabnikovih zasebnih podatkov prek spletnega brskalnika.

Z implementiranim pajkom smo izvedli večji sprehod, ki je zaobjel okoli 100.000 spletnih aplikacij, in zgradili vzorčno podatkovno bazo, ki bo služila nadaljnjim raziskavam na tem področju. Uspešno smo razvili upravljalno orodje za nadzor spletnega pajka. Z njim si lahko ogledamo vse podrobnosti o izvajanju spletnih aplikacij, prenesenih virih in povezovanju aplikacij med seboj ter z zunanjimi ponudniki. Orodje za upravljanje spletnega pajka omogoča ogled zgrajene hierarhije domen, ki jo pajek gradi iz spletnih naslovov preiskanih spletnih aplikacij, in nekaj ogledov analiz nad vsemi podatki v podatkovni bazi. Uspešno smo realizirali tudi

nadzorovanje spletnega pajka prek upravljalnega orodja.

Pridobljena vzorčna podatkovna baza in spletni pajek omogočata nadaljne raziskave. Raziskujemo lahko, ali gostitelji virov prilagajajo vsebino virov glede na spletno aplikacijo, ki vir zahteva. Če se viri razlikujejo, se lahko raziskuje, kako spremembe vplivajo na delovanje spletnih aplikacij in kako posegajo v uporabnikov spletni brskalnik. Že med samim razvojem spletnega pajka je bilo možno opaziti razlike v velikosti enakega vira celo znotraj iste spletne aplikacije, kar potrjuje obstoj omenjenega problema. Izredno pomembna je raziskava o obstoju zajemanja in deljenja uporabnikovih zasebnih podatkov prek izvajalnih datotek, ki se prenesejo prek zunanjih ponudnikov. Sam uporabnik to težko nadzoruje in se niti ne zaveda potencialne ogroženosti. Rešitev diplomske naloge omogoča uporabo spletnega pajka za sprehajanje po dinamičnih spletnih aplikacijah z izbranimi naslovi in za pridobivanje za raziskovalca zanimive podatke.

Sam spletni pajek se sicer še vedno dopolnjuje in izpopolnjuje, saj je različnih spletnih aplikacij res veliko in je skoraj nemogoče predvideti vse možne situacije. Možnosti v nadgradnji spletnega pajka je še veliko. V nadaljnjem razvoju bi bilo potrebno dodelati sistem za prevzem spletnih naslovov, ki trenutno deluje na enostavni vrsti FIFO. Dodalo bi se lahko veliko omejitev, s katerimi bi se izboljšalo zajemanje podatkov, pridobivanje rezultatov in usmerjanje delovanja spletnega pajka. Mednje spada dodajanje omejitve preiskanega števila strani na domeno, omejitev prenašanih parametrov v spletnih naslovih in podobno.

Uporabljeno orodje za izvajanje dinamičnih spletnih aplikacij v peskovniku, PhantomJS, je sicer odlično za testiranje spletnih aplikacij, ker pa gre za projekt, na katerem dela manjše število ljudi, se je izkazalo za zelo nestabilno orodje pri uporabi v spletnem pajku. V samem diplomskem delu je bilo omenjenih nekaj problemov, s katerimi smo se srečali (nestabilnost, počasnost, nepojasnjeno stalno povečevanje porabe pomnilnika). V nadaljnjem razvoju spletnega pajka bi bilo potrebno premisliti o ustreznosti tega orodja in o morebitni zamenjavi.

Največ možnosti za izboljšanje delovanja pajka je na strani podatkovne baze. Kmalu v fazi razvoja je bilo ugotovljeno, da je to največji problem v zmogljivosti pajka. Hitrost podatkovne baze je možno občutno pohitriti pri delovanju na diskih SSD (`Solid-State Drive`). Opaziti je bilo mogoče, da zmogljivost podatkovne

baze hitro pada z naraščanjem števila dokumentov v njej. Nekaj optimizacij je možnih v izboljšanju shranjevanja spletnih naslovov. Uporabljeni spletni naslovi bi se lahko iz vrste odstranjevali, povezave, najdene na spletni aplikacije, pa shranjevale na drugačen način. Še več optimizacij je možnih na področju shranjevanja virov. Trenutno se v podatkovno bazo shrani vsak dobljen vir, zato je njihovo število pri vzorčni bazi hitro preseglo dva milijona dokumentov. Shranjevalo bi se lahko vire glede na združevanje več njegovih atributov, ki bi morali biti unikatni. Nov vir bi se shranil le ob unikatni kombinaciji vrednosti izbranih atributov.

Iz pridobljenih rezultatov vzorčnega sprehoda se je potrdilo, da gre pri povezovanju spletnih aplikacij za zelo aktualen problem. Pokazalo se je namreč, da je stopnja integracije dinamičnih spletnih aplikacij izredno velika. Zelo zanimivo je opažanje, da ponudniki spreminjajo vire. To je za uporabnika zelo zaskrbljujoče, saj lahko ponudnik uporabnikom brez nadzora vsili kodo, ki deluje prek osnovnega namena spletne aplikacije in bi jo lahko označili celo za zlonamerno kodo. Spletni pajek tako omogoča izvajanje kompleksnih metod za analizo povezavovanj ter interakcij med spletnimi aplikacijami in zunanjimi ponudniki. Diplomsko delo predstavlja platformo za izvajanje kompleksnejših raziskovalnih metod in potrjuje, da so takšne raziskave zelo aktualne in smiselne.

Literatura

[1] Bootstrap. Dostopno na: `http://getbootstrap.com/`. 2014, avgust.

[2] Express - node.js web application framework. Dostopno na: `http://expressjs.com/`. 2014, avgust.

[3] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[5] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. *Node. js in Action*. 2014.

[6] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[8] D. Flanagan. *JavaScript: The Definitive Guide*. Definitive Guides. O'Reilly, 2002.

[9] Google. AngularJS: Developer Guide: Introduction. Dostopno na: `https://docs.angularjs.org/guide/introduction`. 2014, avgust.

[10] Jing Han, E. Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366, Oct 2011.

[11] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[12] Ariya Hidayat. PhantomJS | PhantomJS. Dostopno na: `http://phantomjs.org/`. 2014, avgust.

[13] Ariya Hidayat. Web Page Module | PhantomJS. Dostopno na: `http://phantomjs.org/api/webpage/`. 2014, avgust.

[14] Apple Inc. The WebKit Open Source Project. Dostopno na: `https://www.webkit.org/`. 2014, avgust.

[15] Joyent Inc. node.js. Dostopno na: `http://nodejs.org/`. 2014, avgust.

[16] MongoDB Inc. The MongoDB 2.6 Manual - MongoDB Manual 2.6.4. Dostopno na: `http://docs.mongodb.org/manual/`. 2014, avgust.

[17] Matt Sergeant. How does the Internet work - W3C Wiki. Dostopno na: `http://www.w3.org/wiki/How_does_the_Internet_work#Static_vs._Dynamic_Web_Sites`. 2014, september.

[18] Matt Sergeant. node-phantom-simple. Dostopno na: `https://www.npmjs.org/package/node-phantom-simple`. 2014, avgust.

[19] Leon Shklar and Richard Rosen. Web application architecture. *Principles, Protocols and Practices, Editura Wiley*, 2009.

[20] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *Internet Computing, IEEE*, 14(6):80–83, Nov 2010.